САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ФАКУЛЬТЕТ ПМ-ПУ

КАФЕДРА КОМПЬЮТЕРНОГО МОДЕЛИРОВАНИЯ И
МНОГОПРОЦЕССОРНЫХ СИСТЕМ

Федосеев Георгий Александрович

РАЗРАБОТКА СИСТЕМЫ РАСПОЗНАВАНИЯ РЕЧИ ДЛЯ ИНДЕКСИРОВАНИЯ И ПОИСКА В БОЛЬШОЙ КОЛЛЕКЦИИ МЕДИАФАЙЛОВ

Выпускная квалификационная работа

Направление: 03.01.02 «Прикладная математика и информатика»

Научный руководитель:
доктор технических наук, доцент
Дегтярев Александр Борисович

Санкт-Петербург — 2018

Оглавление

Стр.

Введение

За последние несколько лет отрасль машинного распознавания речи развивалась стремительно. Крупные компании, такие как Google, Microsoft и Яндекс, достигли [43; 48; 49] на сегодняшний день уровня 94-95% верного распознавания слов в речи, сокращая до нескольких процентов разрыв с человеческим уровнем, оцениваемым в 96% [20].

С развитием технологий в области нейронных сетей и производства графических ускорителей, позволивших значительно увеличить скорость параллельных вычислений [8], появились проекты с использованием так называемых нейронных сетей глубокого обучения (Deep Neural Networks или DNN) [11; 14; 34; 46; 47]. В таких сетях, по сравнению с классическими, увеличено количество скрытых слоев. Увеличение числа скрытых слоев позволило добиться значительного прогресса во многих задачах, считавшихся ранее доступными только для человека. Не исключением стала и задача распознавания речи.

Решения с использованием нейронных сетей для распознавания речи приходят на смену зарекомендовавшим себя реализациям с использованием скрытых марковских моделей (Hidden Markov Models, HMM) [41]. Решения, использующие HMM, требуют большего вмешательства человека [12] в отличие от более современных методов, использующих нейронные сети. Появляются гибридные системы DNN-HMM, использующие нейронную сеть для предобработки в виде классификатора признаков для уменьшения размерности задачи для HMM [30; 51]. Кроме того, в последнее время решения на основе DNN начали превосходить HMM по точности распознавания [14]. Особенно популярны в последнее время реализации, использующие нейронные сети от «начала и до конца» (End-to-End) [15; 16; 25]. Такие решения минимизируют участие человека в обучении модели, требуя от него лишь определения архитектуры нейронной сети и подготовки данных для тренировки.

Несмотря на значительный прогресс и адаптацию вышеприведенных методов в коммерческих продуктах, в сфере распознавания речи на сегодняшний день существует дефицит систем с открытым исходным кодом, способных до-

стичь показателей, близких к человеческим. Особенно мало систем с поддержкой русского языка [1].

В данной работе рассмотрен процесс создания системы распознавания русской речи с использованием глубоких нейронных сетей, а также применения этой системы для текстового поиска по речи в большой коллекции медиафайлов. Описан принцип работы системы распознавания речи, основанной на использовании глубоких нейронных сетей и применении метода CTC (Connectionist Temporal Classification) [12]. Также исследованы и реализованы способы получения наборов данных для обучения нейронной сети из различных источников, проведено обучение нейронной сети на полученных данных с вариацией гиперпараметров и оценена точность получаемой модели. В конце работы приведена реализация примера возможного использования системы распознавания речи для поиска по речи в коллекции медиа-файлов.

За основу системы распознавания речи взят проект с отрытым исходным кодом Mozilla DeepSpeech по распознаванию английской речи с использованием DNN. Этот проект был начат в мае 2016 года и в ноябре 2017 достиг наименьшего уровня ошибки WER (Word Error Rate) в 6.5% на наборе данных LibriSpeech test-clean [3]. Проект основывается на работе группы исследователей из Baidu Research [16].

Постановка задачи

Целями данной работы являются:

- разработка метода автоматического сбора корпусов транскрибированной русской речи, пригодных для обучения глубоких нейронных сетей, объемом более 500 часов;
- создание системы распознавания русской речи на основе архитектуры DeepSpeech при помощи обучения рекуррентной DNN на примерах *(аудио, транскрипт)* с использованием метода CTC;
- создание системы полнотекстового поиска по речи в большой коллекции медиафайлов.

Система распознавания речи должна распознавать отрывки на русском языке в наборе данных чистой речи средней протяженностью 5-15 секунд со средней ошибкой WER менее 30%.

Система поиска по коллекции медиафайлов должна реализовывать возможность поиска по речи, звучащей в медиафайлах, и позволять перейти к месту звучания искомого отрывка для дальнейшего просмотра или прослушивания.

Для оценки точности распознавания речи используются две метрики:

- **Word Error Rate (WER)** - среднее по совокупности примеров нормализованное расстояние Левенштейна между двумя последовательностями слов. Для каждого примера вычисляется по формуле:

$$WER = \frac{I + D + S}{N} \tag{1}$$

  где I - число добавленных в исходную строку слов, D - число удаленных из исходной строки слов, S - число замененных слов, а N - число слов в исходной строке.

  Эта метрика используется для финальной оценки качества модели на уровне слов.

– **Character Error Rate (CER)** - среднее по совокупности примеров нормализованное расстояние Левенштейна на символьном уровне. Вычисляется аналогично WER, но для двух последовательностей символов, а не слов.

Более приоритетной метрикой для минимизации является WER.

Стоит отметить, что на значение вышеприведенных метрик высокое влияние имеет набор данных, на котором была оценена система распознавания речи. Так на наборе с зашумленной речью значения WER и CER будут выше, чем на наборе с более чистой речью. Поэтому при оценке точности будет указываться набор данных, на котором был получен результат.

Обзор литературы

Рассмотрим наиболее важные публикации о развитии методов применения глубоких рекуррентных нейронных сетей для распознавания временных последовательностей, ведущих к современным End-to-End методам распознавания речи.

В задачах распознавания последовательностей данных используют рекуррентные нейронные сети (Recurrent Neural Networks, RNN) [2; 4; 25; 36; 42]. RNN позволяет использовать информацию о ранее предсказанных элементах последовательности для предсказания следующего. Например, в задаче распознавания речи использование символов, предсказанных несколько шагов назад, помогает нейронной сети определить более подходящий символ для текущего кадра звука в контексте уже предсказанных символов.

Для глубоких нейронных сетей характерна проблема нестабильного градиента, когда нормы градиента экспоненциально растут или убывают от выходного слоя нейронной сети к первому, что соответственно называют проблемой взрывающегося (exploding) и исчезающего (vanishing) градиента. Проблему взрывающегося градиента решают простым ограничением модуля градиента (gradient clipping), как описано в работе [40]. Проблема исчезающего градиента более сложна и особенно критична для RNN, так как количество внутренних слоев зависит от длины последовательности и влияние памяти о прошлых предсказаниях в наивных RNN исчезает уже через несколько шагов времени, что не позволяет использовать более общий контекст для предсказания. Для проблемы исчезающего градиента было предложено несколько решений: использование функции активации ReLU [35], GRU (Gated Recurrent Unit) [39], LSTM (Long Short Term Memory) [53]. Последние два метода исследованы и сравнены в работе [18]. Метод использования LSTM ячеек памяти является наиболее успешным на сегодняшний день и получил наибольшее распространение.

LSTM подход заменяет простые функции активации в нейроне (такие как логистическая функция или гиперболический тангенс) ячейками с памятью, которые могут запоминать аналоговое значение. Каждая такая ячейка памяти имеет затворы (gates) на входе и выходе, которые контролируют, может

ли входной сигнал изменить внутреннюю память и может ли внутренняя память повлиять на выходной результат соответственно. Эти затворы оперируют на основе своих собственных обучающихся весов, соответствующих входному сигналу и LSTM ячейке на прошлом шаге. Также существует «забывающий» затвор (forget gate), который контролирует убывание значения, хранящегося в памяти ячейки. Таким образом, пока входной и забывающий затвор закрыты, значение градиента хранящееся в памяти остается неизменным во времени.

LSTM стали активно использовать после успеха в распознавании последовательностей рукописного текста, описанного в работе [4]. После этого данный подход был успешно применен в машинном переводе текста [5; 9; 22], генерации рукописного текста [23], текстовом описании изображений [33; 45], предсказании вывода простых компьютерных программ [52], распознавании речи [21; 25; 26] и других областях связанных с обработкой последовательностей.

Прорывным подходом, на базе которого построено множество современных End-to-End систем распознавания речи, стал подход Connectionist Temporal Classification (CTC), предложенный исследователем Alex Graves в статье [12]. В работе описан вид выходного слоя нейронной сети, а также функция потерь, вычисляемая на основе значений этого слоя. CTC не ограничивает выбор архитектуры сети. Подход развит в работе [24, с. 52]. CTC решает проблему необходимости точной разметки данных для задач распознавания последовательностей. В сфере распознавания речи эта проблема состоит в сопоставлении последовательности кадров аудио и последовательности букв - одной букве может соответствовать множество кадров аудио. CTC метод также предлагает дифференцируемую функцию потерь CTC Loss, применимую для обучения методом стохастического градиентного спуска. Подробное описание подхода приведено в главе 2 настоящей работы.

Альтернативой CTC для End-to-End распознавания речи являются набирающие популярность модели Sequence to Sequence (Sec2Sec) со вниманием (Attention) [7; 19]. Такие модели состоят из кодировщика и декодировщика. Кодировщик сжимает информацию кадров аудио в более компактное векторное представление с помощью уменьшения количества нейронов от слоя к слою, а декодировщик на основе этого сжатого представления и рекуррентной нейронной сети восстанавливает последовательность символов, фонем или даже слов. Механизм внимания реализован в декодировщике, который фокусируется

более всего только на части сжатой информации, выбирая область для более «внимательного» рассмотрения основываясь на контексте уже предсказанной последовательности.

Также существуют и гибридные подходы CTC+Attention [6; 31], заявляющие улучшение распознавания по сравнению с CTC и Attention в отдельности.

Архитектура DeepSpeech, описанная в публикации [16] является примером End-to-End DNN, что предполагает более простой процесс подготовки данных для обучения, отсутствие необходимости ввода таких понятий как фонема, а также минимальной подстройки параметров. Данная архитектура включает несколько скрытых слоев, один из которых является рекуррентным с LSTM ячейками, на остальных используется функция активации ReLU. В качестве функции потерь используется CTC Loss. Архитектура оптимизирована для обучения сразу на нескольких графических ускорителях. Кроме этого для корректировки предсказаний используется языковая модель, способная оценивать последовательности слов. В работе заявлено достижения 16% WER на полном тестовом поднаборе набора данных «Switchboard Hub5'00», состоящем из 40 телефонных разговоров на английском языке.

Развитием DeepSpeech стала архитектура DeepSpeech 2 [15]. В работе оптимизируется архитектура для использования ее в промышленном распознавании речи. Также описываются незначительные изменения, позволившие перенести систему с обучения английской речи на обучение севернокитайской (мандаринской) речи. В работе завялено достижение минимального WER в 3.1% для набора данных чтения журналов «WSJ eval'92» и 21.59% для набора данных зашумленной речи «CHiME eval real», тогда как человеческие показатели оценены в 5.03% и 11.84% WER соответственно.

В статье [10] описана архитектура Cold Fusion, основанная на подходе Seq2Seq со вниманием. В работе предложено использовать языковую модель не только для декодирования результата нейронной сети, но и для ее обучения. В работе заявлено достижение 27.5% WER на собственном наборе данных.

На основе рассмотренных работ можно заключить, что самыми активно применяемыми и перспективными решениями для End-to-End распознавания речи на данный момент являются решения с использованием CTC и Sec2Sec+Attention.

Глава 1
Выбор инструментов для реализации задачи

1.1 Выбор аппаратного обеспечения

Так как реализация поставленной задачи включает в себя использование глубоких нейронных сетей для распознавания речи, то требования к аппаратному обеспечению реализации достаточно высоки. Для обучения нейронных сетей с несколькими внутренними слоями с тысячами нейронами в каждом за обозримое время используют графические ускорители. Также необходимым условием является наличие достаточного дискового пространства на используемой ЭВМ для хранения гигабайт данных для обучения нейронной сети.

Исходя из вышеперечисленных требований, для реализации поставленной задачи была использована ЭВМ ресурсного центра «Вычислительный центр» СПбГУ со следующими характеристиками:

- **ЦПУ:** 2 x Intel(R) Xeon(R) CPU E5-2690 v4 @ 2.60 ГГц
- **ОЗУ:** 256 ГБ
- **Графические процессоры:** 2 x Nvidia Tesla P100, по 16 ГБ видеопамяти

1.2 Выбор базовой архитектуры для построения системы распознавания речи

Для реализации системы распознавания речи в данной работе была выбрана существующая архитектура глубокой нейронной сети для распознавания речи Mozilla DeepSpeech. Реализация данной архитектуры доступна в виде открытого исходного кода на сайте GitHub и была в свою очередь построена на основе статьи исследователей из Baidu Research об архитектуре DeepSpeech [16]. Схема архитектуры DeepSpeech представлена на рисунке 1.1. Для описания структуры нейронной сети и оптимизации вычислений в проекте Mozilla

DeepSpeech использован фреймворк для машинного обучения TensorFlow. Данный фреймворк позволяет «замораживать» (freeze) обученный граф нейронной сети и использовать модель на других устройствах (в том числе и мобильных). На момент написания данной работы, проект Mozilla DeepSpeech имел самую большую популярность и активность среди проектов по распознаванию речи с открытым исходным кодом по данным сайта GitHub. Данный проект нацелен на распознавание английской речи и добился на данный момент результата ниже 10% WER. Проект Mozilla DeepSpeech также не привязан к конкретному языку и имеет возможность адаптирования под русский язык.

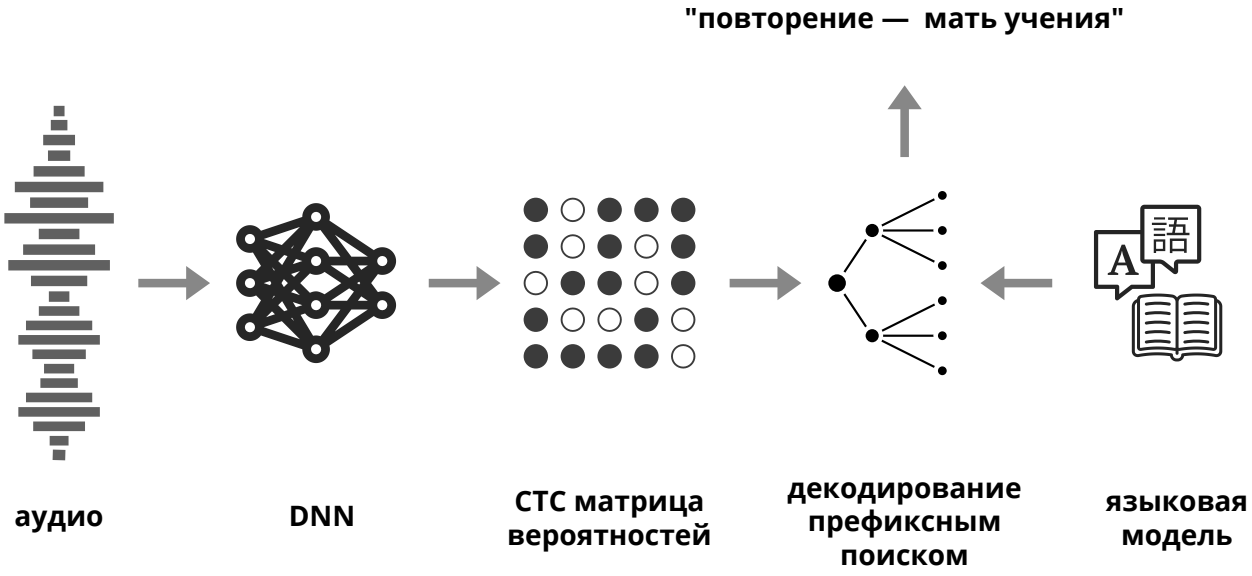


Рис. 1.1 — Архитектура DeepSpeech

1.3 Выбор источников данных для обучения акустической модели

Для обучения глубокой нейронной сети распознаванию речи необходимо 500-2000 часов речи с соответствующими транскриптами. Было рассмотрено несколько источников данных русской речи.

1.3.1 Набор речевых данных с сайта VoxForge.org

Существует готовый набор транскрибированных речевых данных на сайте voxforge.org, созданный сообществом пользователей, которые записали отрывки из русских литературных произведений, прочитанные своим голосом. Объем этих данных составляет около 26 часов речи. Данный источник, в отличие от остальных, рассмотренных ниже, предоставляет готовый набор данных.

1.3.2 Видео-хостинг YouTube

Другим источником транскрибированной речи явился сайт видео-хостинга YouTube, где для видео существуют субтитры. Предоставляются два вида субтитров: созданные человеком субтитры и субтитры автоматически распознанные с помощью алгоритмов ASR от Google. Оба вида имеют свои плюсы и минусы.

Созданные человеком субтитры имеют большую точность по смыслу транскрипта, но меньшую точность по соотнесению текста и аудио во времени. Также плюсом автоматических субтитров является большее количество видео, которые имеют эти субтитры. Автоматические субтитры доступны почти в любом видео, где можно распознать речь. Кроме большей точности по времени и объема доступных данных, плюс автоматических субтитров состоит в том, что из них можно извлечь информацию о времени начала и конца каждого из распознанных слов.

На видео-хостинге YouTube представлены миллионы видео с русской речью, записанной различными голосами, с помощью разнообразных записывающих устройств и в самых различных шумовых условиях. Обучение на таком разнообразии данных при достаточном качестве транскриптов положительно сказывается на адаптации нейронной сети к различным условиям записи и акцентам [38]. Из-за объема доступной информации и разнообразия речи, видеохостинг YouTube был использован как основной источник речевых данных.

1.3.3 Записи радио-передач

Еще одним потенциальным источником речевых данных стали записи и транскрипты радио-передач на сайте «Эхо Москвы». В отличие от предыдущих источников, для подготовки данных из этого источника необходимо больше усилий, так как аудио и текст представлены отдельно друг от друга и не выровнены по времени, т.е. для текстового отрывка неизвестны границы временного отрезка в аудио-файле. Выравнивание текста по аудио в автоматическом режиме описано в параграфе 3.4.

1.3.4 Аудио-книги

В сети Интернет распространены аудио-версии популярных книг. Сопоставляя эти аудио-записи с текстом соответствующих книг, можно создать набор транскрибированных речевых данных. Как и с радио-передачами для использования речевых данных аудио-книг необходим предварительный этап выравнивания текста по аудио. Плюсом данного источника является объем доступных данных.

1.3.5 Фильмы с субтитрами

Транскрибированные речевые данный можно извлечь из фильмов с субтитрами. Данный источник можно сравнить с источником YouTube из 1.3.2. Недостатком является меньшая средняя плотность речи на единицу объема данных. В условиях ограниченности вычислительных ресурсов, скорости скачивания и дискового пространства, YouTube является более оптимальным источником из-за большей плотности речи и возможности загрузки только аудио соответствующего видео.

1.3.6 Выбранные источники данных транскрибированной речи

В качестве оптимальных источников для автоматического создания набора речевых данных из описанных выше были выбраны: видео-хостинг YouTube, сайт VoxForge.org и сайт радио-станции «Эхо Москвы».

1.4 Выбор языка программирования

Для реализации компонент системы, описанной в постановке задачи был выбран скриптовый язык Python версии 2.7. Выбор обосновывается наличием большого числа библиотек кода для машинного обучения, а также для обработки аудио. Также этот язык используется для описания акустической модели, выбранной в параграфе 1.2. Версия 2.7 языка Python была выбрана из-за бóльшего числа библиотек, совместимых с ней.

Глава 2
Структура и принцип работы акустической модели

В данной главе рассмотрена структура и принцип работы акустической модели, предсказывающей последовательность символов алфавита для входного аудио сигнала.

Введем множество символов (алфавит) A длины $[A]$, состоящий из строчных букв рассматриваемого языка и пробела. Также рассмотрим алфавит $A' = A \cup \{пропуск\}$ длины $[A'] = n$. Символ *пропуск* необходим для работы CTC метода.

Рассмотрим входную последовательность длины T векторов $\mathbf{x}$. Эта последовательность состоит из векторов признаков, каждый из которых получен извлечением из аудио сигнала короткой длины (обычно 20 миллисекунд) мелкепстральных коэффициентов (MFCC). Выделяемые MFCC признаки численно описывают отрезок аудио и представляют собой данные, на основе которых будут предсказываться символы. При извлечении алгоритмом MFCC можно задать количество признаков, пусть оно равно m.

Определим также рекуррентную нейронную сеть с m входами и n выходами и с вектором весов ω. Вектор весов будет определять отображение N_ω, переводящее входную последовательность $\mathbf{x}$ в последовательность векторов длины n.

$$N_\omega : (\mathbb{R}^m)^T \longrightarrow (\mathbb{R}^n)^T \tag{2.1}$$

2.1 Структура нейронной сети

Используемая нейронная сеть состоит из 5 скрытых слоев. Входной слой имеет m входов, соответствующих m MFCC частотным признакам, извлекаемым из короткого фрагмента аудио. Далее идут еще 2 подобных слоя. В первых 3-х слоях нейроны между соседними слоями соединены всеми возможными способами и используют функцию активации ReLU (Rectifier Linear Unit). Четвертый слой является двунаправленным рекуррентным слоем (Bidirectional

Recurrent layer) с LSTM ячейками и гиперболическим тангенсом в качестве функции активации. Далее результат подается в пятый слой с ReLU активацией. После этого идет выходной слой размера $n = [A']$, где значение на выходе каждого из нейронов пропорционально вероятности соответствующей буквы алфавита.

Функция активации ReLU определяется как:

$$ReLU(x) = max(0, x) \tag{2.2}$$

Ко всем пяти слоям применяется dropout с вероятностью исключения 0.3. Применение dropout к слою нейронной сети обнуляет значение нейронов в слое с вероятностью $1 - P_{keep}$. Это помогает предотвратить переобучение нейронной сети. Обычно используют значение $P_{keep} = 0.7$, таким образом из дальнейших вычислений исключается 30% нейронов данного слоя.

Используется только один рекуррентный слой с LSTM ячейками, так как рекуррентные слои труднее вычислять параллельно на графических ускорителях и использование большего числа подобных слоев существенно замедлило бы как процесс обучения, так и предсказание. Рекуррентный слой расположен ближе к выходу так как при прохождении данными слоев увеличивается уровень абстракции и на этом уровне сети с бóльшим уровнем абстракции применение рекуррентного слоя наиболее оптимально.

Иллюстрацию работы рекуррентной сети с обратным ходом можно увидеть на рисунке 2.1.

2.2 Описание процесса обучения акустической модели

Для обучения нейронной сети используются пары *(аудио, транскрипт)*. Весь набор данных для обучения разбит на 3 поднабора: набор для тренировки (train set), набор для проверки (dev set) и набор для тестирования (test set). Наборы train, dev и test не должны пересекаться и обычно получаются из исходного набора данных путем разделения его в пропорции 60:20:20 соответственно. Набор train используется для корректировки весов нейронной сети в процессе орбучения. Набор dev используется для проверки точности нейронной сети в

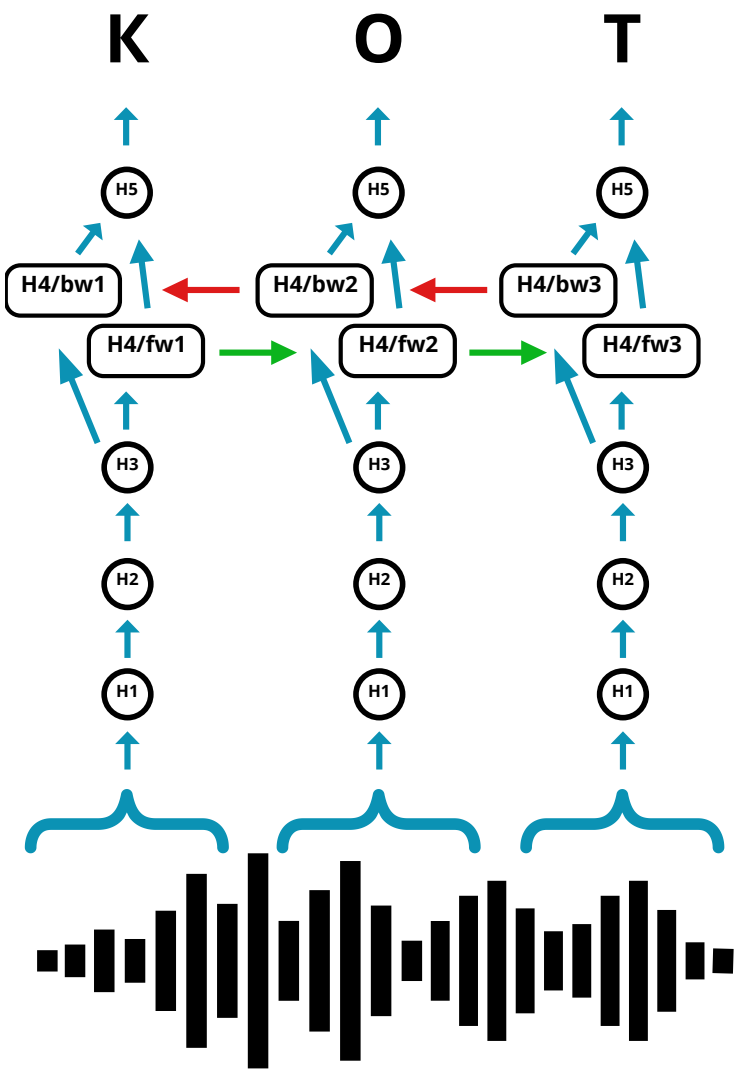


Рис. 2.1 — Путь данных через нейронную сеть от аудио сигнала до предсказанных символов. Четвертый слой является рекуррентным с обратным ходом и использует LSTM ячейки памяти для хранения данных о предыдущих предсказаниях.

процессе обучения на отличных от train данных, так как нейронная сеть может переобучиться и адаптироваться именно примерам из train. Набор test используется для финальной проверки вконце обучения, на этом наборе вычисляется как среднее значение CTC Loss, так и среднее значение WER по всем примерам для определения точности обученой модели на уровне слов.

Обучение нейронной сети происходит по следующему алгоритму:

1. На входной слой нейронной сети подаются вектора признаков MFCC, извлеченные из аудио.
2. Данные, проходя через нейронную сеть и преобразуемые с помощью весов нейронной сети, результируют в виде предсказанной матрицы символов, где каждый столбец является распределением вероятности по символам алфавита A' в момент времени t.

3. На основе предсказанной матрицы и ожидаемого транскрипта вычисляется значение ошибки CTC Loss для данного примера. Исходя из вычисленного значения происходит корректировка весов ω нейронной сети N методом Adam [32], являющимся модификацией метода стохастического градиентного спуска.
4. Процесс повторяется до тех пор пока значение ошибки на проверочном наборе данных (validation set) продолжает снижаться.

Таким образом итеративно происходит улучшение точности акустической модели.

Стоит отметить, что для ускорения вычислений на графических процессорах путем параллелизации процесса примеры из наборов train, dev и test подаются группами или батчами (batch). Обновление весов нейронной сети происходит на основе среднего значения CTC Loss по группе примеров.

2.3 Функция потерь CTC Loss для обучения нейронной сети

В 2006 году Алекс Грейвс (Alex Graves) предложил использовать функцию CTC Loss для обучения RNN распознаванию последовательностей [12].

Рассмотрим выходную последовательность $\mathbf{y} = N_\omega(\mathbf{x})$. Каждый элемент этой последовательности является вектором распределения вероятностей по каждому из символов алфавита A' в момент времени t. Таким образом элемент y_k^t - это вероятность того, что в момент времени t во входной последовательности произнесен символ k из алфавита A'.

Пример выходной последовательности $\mathbf{y}$ в виде матрицы, где каждый элемент обозначает уверенность акустической модели в определенном символе на данном шаге времени, представлен ниже на рисунке 2.2.

Рассмотрим последовательности длинны T, где каждый элемент является номером из $[1, n]$, который соответствует символу в алфавите A'. Назовём такие последовательности *путями* и будем обозначать как π. Любой путь можно преобразовать в последовательность символов, произведя замену номера на соответствующий символ. Рассмотрим различные пути в предсказанной нейрон-

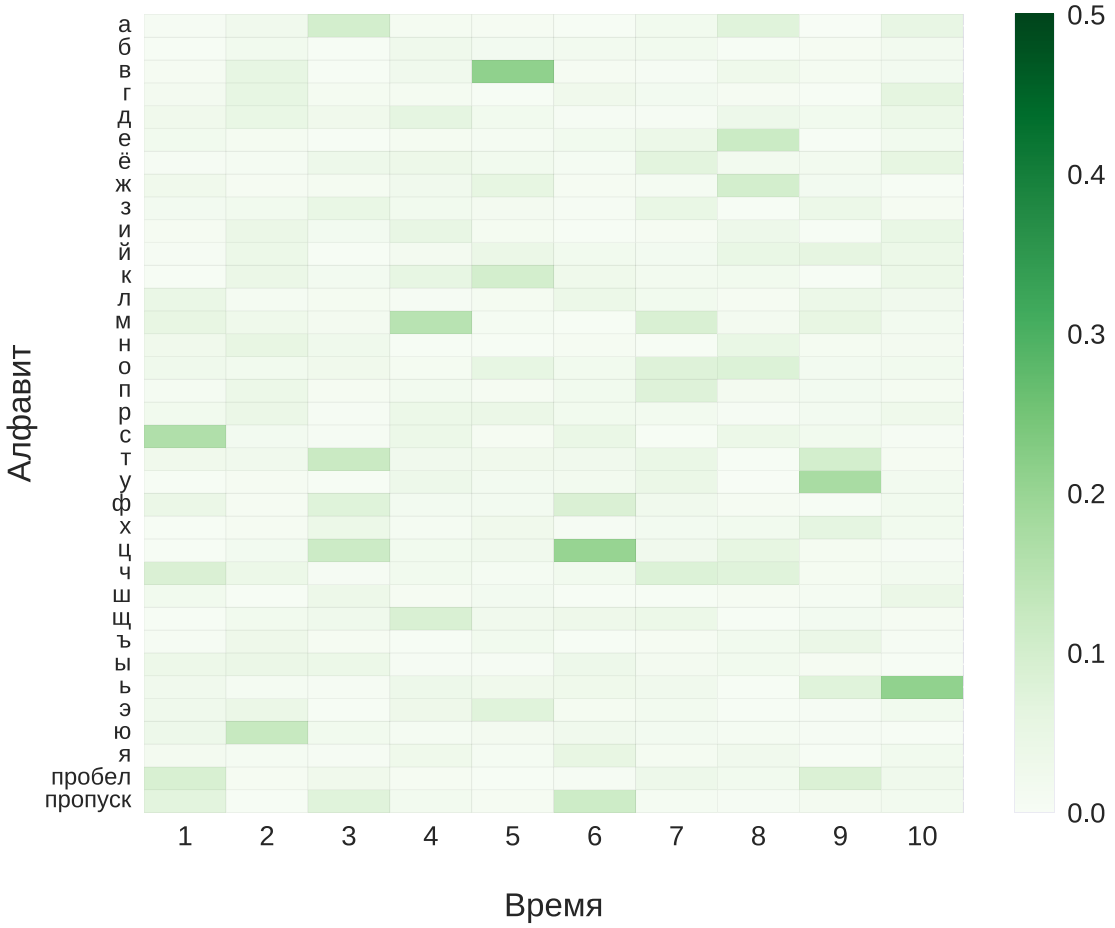


Рис. 2.2 — Предсказанная акустической моделью матрица. Каждый столбец содержит распределение вероятностей по символам расширенного алфавита A' (сумма элементов в столбце равна 1). Чем больше значение элемента матрицы, тем больше акустическая модель уверена в символе, соответствующем строке матрицы, на шаге времени, соответствующем столбцу.

ной сетью матрице. Таких путей может быть T^n штук. И вероятность каждого путя при данном $\mathbf{x}$ может быть вычислена по формуле:

$$p(\pi|\mathbf{x}) = \prod_{t=1}^{T} y^t_{\pi_t}, \forall \pi \in A'^T \tag{2.3}$$

При распознавании речи ожидается, что каждой сказанной букве будет соответствовать один символ в выходной последовательности, а промежутки между словами преобразуются в единичный знак пробела. Но так как в аудио некоторые звуки, соответствующие одному символу, могут растянуться на период времени, более долгий чем наше окно в 20 миллисекунд, в матрице предсказаний одному символу, ожидаемому в выходной последовательности, может соответствовать несколько столбцов.

Поэтому, чтобы получить финальную распознанную строку текста, необходимо преобразовать путь. Рассмотрим функцию $B(\pi) = C(D(\pi))$ на множестве путей длины T,с алфавитом из n символов. Функция B сначала склеивает все повторяющиеся в пути символы в один (преобразование D), а затем удаляет символы *пропуска* (преобразование C). Так получается результат функции B. Таким образом B сюръективно: множество путей может соответствовать одному результату функции B. Тут стоит заметить смысл дополнительного символа *пропуск* в алфавите A'. Он необходим для распознавания слов с повторяющимися последовательно символами, чтобы при преобразовании функцией B эти символы не «склеились» в один. Для достижения этого нейронная сеть должна быть обучена таким образом, чтобы ставить символ *пропуска* между словами с удвоенными буквами.

Для обучения нейронной сети с помощью метода обратного распространения ошибки необходима сама функция вычисляющая ошибку предсказания для входной последовательности $\mathbf{x}$, нейронной сети N_ω и ожидаемого (верного) результата l. Также необходимо, чтобы эта функция была дифференцируема по переменным выходного слоя нейронной сети y_k^t. Определим такую функцию ошибки как минус натуральный логарифм от вероятности верного результата:

$$L_{CTC} = -ln(p(l|\mathbf{x})) \tag{2.4}$$

Для получения вероятности результата l необходимо сложить вероятности всех путей, которые приводят к результату l:

$$p(l|\mathbf{x}) = \sum_{\pi \in \mathbb{N}^T : B(\pi) = l} p(\pi|\mathbf{x}) \tag{2.5}$$

Так как нахождение всех путей, ведущих к результату l с помощью функции B перебором всех возможных путей очень затратно, необходим более оптимальный способ. В работе Грейвса предложено использовать модифицированный алгоритм прямого-обратного хода [12]. Этот алгоритм динамического программирования также применяется и в HMM.

Основная идея алгоритма заключается в том, что вероятность результата l может быть вычислена с помощью рекурсивной функции. Рассмотрим построение этой функции.

Для начала, для произвольной последовательности символов q длины r введем подпоследовательности $q_{1:p}$ и $q_{r-p:r}$ состоящие из первых p символов q и последних p символов q соответственно.

Определим последовательность l', полученную из l путем вставки символа пропуска между любыми двумя символами l, а также в начале и в конце l. Таким образом $[l'] = 2[l] + 1$.

Построим таблицу, где каждой строке последовательно будет соответствовать символ из l', а столбцу - шаг $t \in [1, T]$.

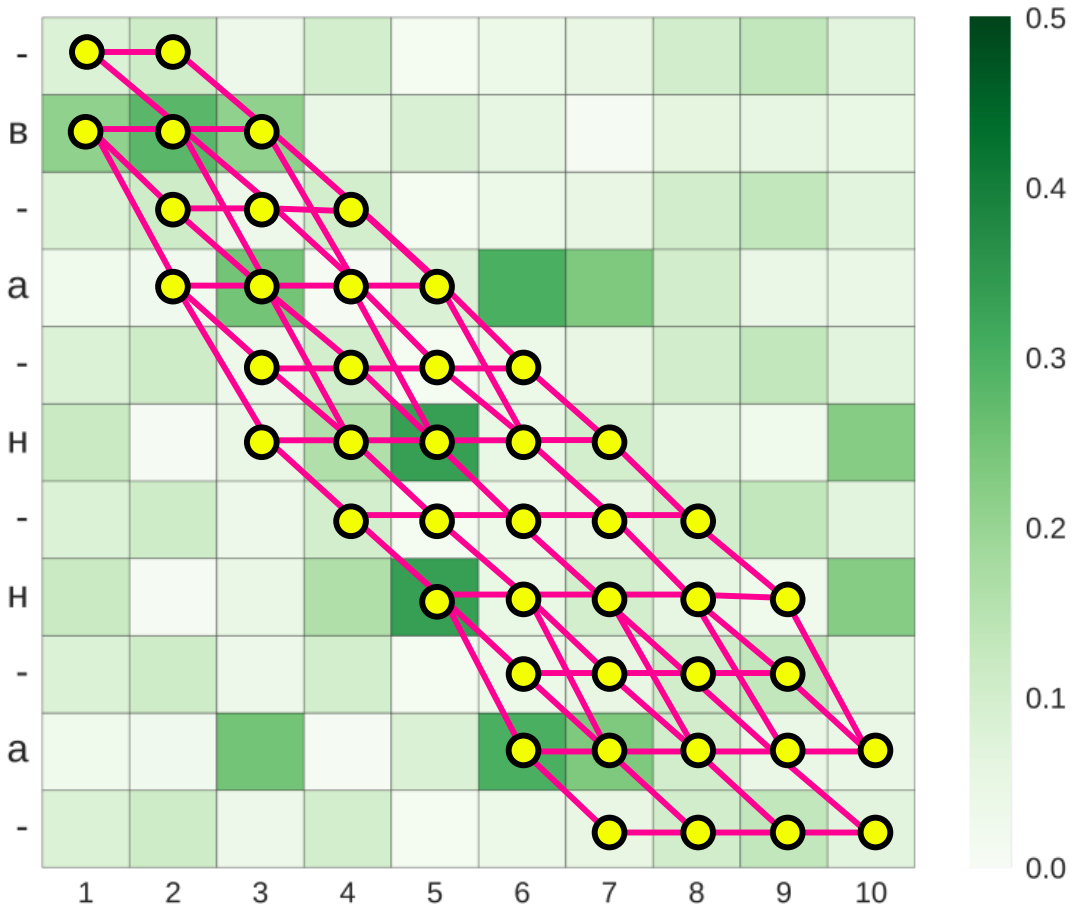


Табл. 2.3 — Всевозможные пути, приводящие за время T к результату l при преобразовании их функцией B.

С помощью таблицы 2.3 рассмотрим различные пути $\overline{\pi}$ длины T, которые могут начинаться с первого символа l или пропуска, а каждый следующий символ может быть либо пропуском, либо следующим символом из l либо повторением текущего символа из l. Также для $\overline{\pi}$ необходимо, чтобы:

$$B(\overline{\pi}) = l \tag{2.6}$$

Для каждого элемента таблицы можно вычислить функцию $\alpha_t(s)$, определяющую вероятность всех путей из таблицы 2.3, начинающихся в первой или второй строке первого столбца и заканчивающихся в ячейке на пересечении строки с номером s и столбца с номером t.

$$\alpha_t(s) \stackrel{def}{=} \sum_{\pi \in \mathbb{N}^T : D(\pi_{1:t}) \in \{l'_{1:s}, l'_{2:s}\}} \prod_{\tau=1}^{t} y^{\tau}_{\pi_\tau}, \tag{2.7}$$

где D - преобразование, склеивающее одинаковые символы, стоящие последовательно, в один.

Но можно заметить, что каждый элемент таблицы $\alpha_t(s)$ можно вычислить, зная элементы $\alpha_{t-1}(s)$, $\alpha_{t-1}(s-1)$ и $\alpha_{t-1}(s-2)$.

Если для первого столбца таблицы мы определим:

$$\alpha_1(1) = y^1_b \tag{2.8}$$

$$\alpha_1(2) = y^1_{l_1} \tag{2.9}$$

$$\alpha_1(s) = 0, \forall s > 2, \tag{2.10}$$

где y^1_b - вероятность символа *пропуска* на первом шаге времени,

то для последующих шагов времени из определения B получаем рекуррентные соотношения:

$$\alpha_t(s) = \begin{cases} (\alpha_{t-1}(s) + \alpha_{t-1}(s-1)) y^t_{l'_s} & \text{если } l'_s = b \text{ или } l'_{s-2} = l'_s \\ (\alpha_{t-1}(s) + \alpha_{t-1}(s-1) + \alpha_{t-1}(s-2)) y^t_{l'_s} & \text{иначе} \end{cases} \tag{2.11}$$

Чтобы выполнялось условие (2.6), необходимо, чтобы путь $\overline{\pi}$ заканчивался символом пропуска либо последним символом из l. Поэтому для вычисления суммы вероятности всех путей, ведущих к результату l нужно вычислить сумму функций альфа на последнем шаге времени от последнего символа l и от символа пропуска (предпоследнего и последнего элемента последовательности l'):

$$p(l|\mathbf{x}) = \alpha_T([l'] - 1) + \alpha_T([l']) \tag{2.12}$$

Таким образом мы получили более эффективный способ вычисления вероятности верного результата l при данной входной последовательности аудио

признаков $\mathbf{x}$ и акустической модели N_ω. И теперь можем применять это для вычисления функции потерь L_{CTC}.

Но для обучения нейронной сети методом обратного распространения ошибки необходимо знать, как вычислять частную производную от функции L_{CTC} по переменным выходного слоя y_k^t:

$$\frac{\partial L_{CTC}}{\partial y_k^t} = -\frac{\partial(ln(p(l|\mathbf{x})))}{\partial y_k^t} = -\frac{1}{p(l|\mathbf{x})}\frac{\partial p(l|\mathbf{x})}{\partial y_k^t} \tag{2.13}$$

Для этого введем еще одну вспомогательную функцию $\beta_t(s)$, которая будет определять суммарную вероятность всех путей, начинающихся в ячейке (s,t) таблицы 2.3 и заканчивающихся в ячейке $([l'],T)$ - на последнем символе l или в ячейке $([l']-1,T)$ - на символе пропуска:

$$\beta_t(s) \overset{def}{=} \sum_{\pi\in\mathbb{N}^T : D(\pi_{t:T})\in\{{l'}_{s:[l']}, {l'}_{s:[l']-1}\}} \prod_{\tau=1}^{t} y_{\pi_\tau}^\tau \tag{2.14}$$

Рассмотрим значения $\beta_t(s)$ для последнего столбца.

$$\beta_T([l']) = y_b^T \tag{2.15}$$

$$\beta_T([l']-1) = y_{l_{[l]}}^T \tag{2.16}$$

$$\beta_T(s) = 0, \forall s < [l']-1, \tag{2.17}$$

где y_b^T - вероятность символа *пропуска* на последнем шаге времени.

Также как и с функцией $\alpha_t(s)$, для $\beta_t(s)$ существуют рекуррентные соотношения, вытекающие из определения функции B, позволяющие вычислить эту функцию для всех ячеек таблицы 2.3:

$$\beta_t(s) = \begin{cases} (\beta_{t+1}(s) + \beta_{t+1}(s+1))y_{l'_s}^t & \text{если } l'_s = b \text{ или } l'_{s+2} = l'_s \\ (\beta_{t+1}(s) + \beta_{t+1}(s+1) + \beta_{t+1}(s+2))y_{l'_s}^t & \text{иначе} \end{cases} \tag{2.18}$$

Заметим, что при данном l выражение $\frac{\alpha_t(s)\beta_t(s)}{y_{l'_s}^t}$ в конкретной ячейке (s,t) дает суммарную вероятность всех путей, проходящих на шаге времени t через символ s:

$$\alpha_t(s)\beta_t(s) = \sum_{\pi \in \mathbb{N}^T : B(\pi)=l, \pi_t = l'_s} y^t_{l'_s} \prod_{\tau=1}^{T} y^\tau_{\pi_\tau} \tag{2.19}$$

Отсюда получаем что:

$$\frac{\alpha_t(s)\beta_t(s)}{y^t_{l'_s}} = \sum_{\pi \in \mathbb{N}^T : B(\pi)=l, \pi_t = l'_s} \prod_{\tau=1}^{T} y^\tau_{\pi_\tau} = \sum_{\pi \in \mathbb{N}^T : B(\pi)=l, \pi_t = l'_s} p(\pi|\mathbf{x}) \tag{2.20}$$

Тогда для каждого шага времени t мы можем вычислить

$$p(l|\mathbf{x}) = \sum_{s=1}^{[l']} \frac{\alpha_t(s)\beta_t(s)}{y^t_{l'_s}} \tag{2.21}$$

Возьмем частную производную от этого выражения по переменной выхода нейронной сети, соответствующей символу k на шаге времени t:

$$\frac{\partial p(l|\mathbf{x})}{\partial y^t_k} = -\frac{1}{{y^t_k}^2} \sum_{s : l'_s = k} \alpha_t(s)\beta_t(s) \tag{2.22}$$

И подставляя 2.22 и 2.12 в 2.13, получаем искомую частную производную для функции L_{CTC}:

$$\frac{\partial L_{CTC}}{\partial y^t_k} = \frac{1}{\alpha_T([l'] - 1) + \alpha_T([l'])} \frac{1}{{y^t_k}^2} \sum_{s : l'_s = k} \alpha_t(s)\beta_t(s) \tag{2.23}$$

2.4 Вычисление результата предсказания с помощью максимального декодирования (greedy search)

Самым вероятным результатом исходя из предсказанной матрицы 2.2 будет являться

$$h(\mathbf{x}) = \arg\max_{l \in A^{\leq T}} p(l|\mathbf{x}) \tag{2.24}$$

Но так как число возможных результатов l для перебора слишком высоко мы будем использовать приближенные решения.

Очевидным решением является выбрать путь собранный из элементов с максимальной вероятностью на каждом шаге t в матрице предсказаний:

$$h(\mathbf{x}) \approx B(\arg\max_{\pi \in N^T} p(\pi|\mathbf{x})) \tag{2.25}$$

Иллюстрацию этого решения можно увидеть на рисунке 2.4.

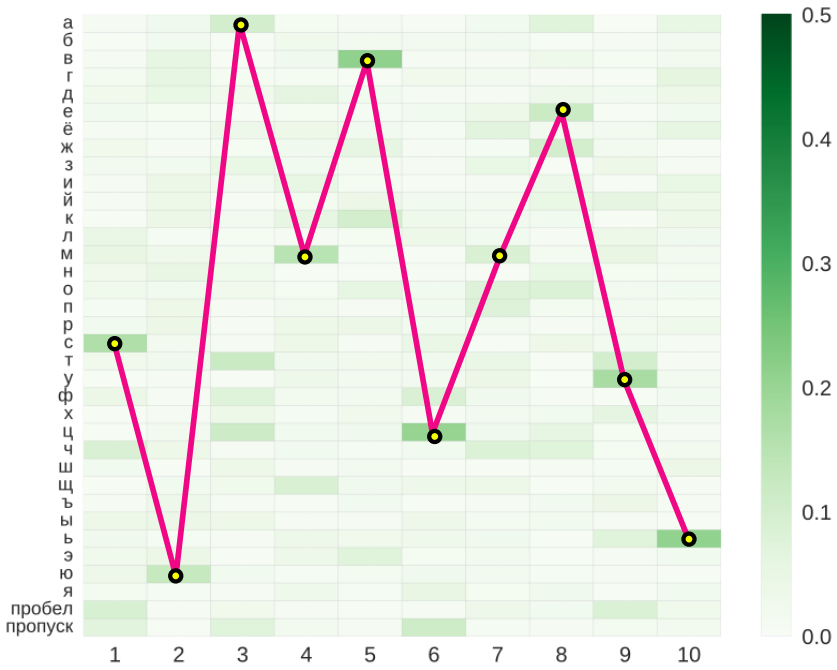


Рис. 2.4 — Наивное решение декодирования CTC матрицы (Greedy Decoding).

Но такое решение не всегда дает наиболее вероятный результат из-за того, что одному и тому же результату может соответствовать несколько путей и вероятность какого-то результата может оказаться больше чем вероятность очевидного решения.

2.5 Вычисление результата предсказания с помощью лучевого поиска (beam search)

Чтобы попытаться найти эти решения используется лучевой поиск (beam search). Также этот поиск называется префиксным, так как на каждом шаге

выбираются лучшие префиксы. Префиксы состоят из символов алфавита A и не могут включать символ *пропуска*.

Рассмотрим алгоритм префиксного поиска [13].

Пусть текущие префиксы шага t алгоритма хранятся в множестве R_t. На шаге t для каждого префикса хранятся 2 вероятности: $P_b(t, l)$ и $P_{nb}(t, l)$.

$P_b(t, l)$ это вероятность равная сумме вероятностей всех путей протяженности t, преобразуемых функцией B в результат l, которые заканчиваются символом *пропуска*.

$P_{nb}(t, l)$ это вероятность равная сумме вероятностей всех путей протяженности t, преобразуемых функцией B в результат l, которые заканчиваются символом алфавита A.

Такое разделение необходимо, чтобы было возможно декодировать повторяющиеся символы алфавита A.

Введем вспомогательные функции $num(c)$, которая превращает символ c в его номер в алфавите A' и функцию $last(s)$, которая возвращает последний элемент произвольной конечной последовательности s.

Также у нас есть значения вероятностей из выходной CTC матрицы, обозначим элементы матрицы как $P_{ctc}(t, num(c))$, где t - номер столбца, а c - символ алфавита A'.

Введем функцию $P_{LM}(x)$, возвращающую вероятность встретить последовательность слов x в речи по оценке языковой модели.

Алгоритм инициализируется с вероятностями $P_b(0, пустота) = 1$ и $P_{nb}(0, пустота) = 0$ и с множеством R_0 состоящим из одного пустого префикса (строки длины 0).

На каждом шаге $t \in [1, T]$ алгоритма для всех префиксов из R_{t-1} происходит перебор символов c из алфавита A':

Если $c =$ ***пропуск***, то вероятность $P_b(t, l)$ не измененного префикса l увеличивается следующим образом:

$$P_b(t, l) \mathrel{+}= P_{ctc}(t, num(пропуск))(P_b(t-1, l) + P_{nb}(t-1, l)) \tag{2.26}$$

Иначе $c \in A$ в этом случае рассматривается префикс $l_1 = l + c$ и возможны 3 ситуации:

1. новый символ c повторяет последний в последовательности l: $c = last(l)$

   Тогда увеличиваем вероятности для случая, когда предыдущим символом в соответствующем пути π был *пропуск* (тогда префикс удлиняется повторяющимся символом) и для случая, когда предыдущим символом был также символ c (тогда символ сливается с предыдущим):

$$P_{nb}(t, l_1) \mathrel{+}= P_{ctc}(t, num(c))P_b(t-1, l) \tag{2.27}$$

$$P_{nb}(t, l) \mathrel{+}= P_{ctc}(t, num(c))P_{nb}(t-1, l) \tag{2.28}$$

2. новый символ c является символом *пробел* или достигнут последний временной шаг: $t = T$

   В этом случае используем языковую модель для оценки текущей получившейся последовательности l_1:

$$P_{nb}(t, l_1) \mathrel{+}= P_{ctc}(t, num(c))(P_{nb}(t-1, l) + P_b(t-1, l))P_{LM}(l_1) \tag{2.29}$$

   Стоит заметить что языковая модель рассматривает последовательность l_1 как последовательность слов, а не символов.

3. новый символ $c \in A \setminus \{\textit{пробел}\}$

   Тогда увеличиваем вероятность текущего префикса l_1, без применения языковой модели:

$$P_{nb}(t, l_1) \mathrel{+}= P_{ctc}(t, num(c))(P_{nb}(t-1, l) + P_b(t-1, l)) \tag{2.30}$$

Также может возникнуть ситуация, когда на шаге $t-1$ префикс l_1 уже возникал, но был отсеян из-за низкой вероятности, то есть его нет в множестве в R_{t-1}. Чтобы использовать вероятности l_1 предыдущего шага выполним следующее:

$$P_{nb}(t, l_1) \mathrel{+}= P_{ctc}(t, num(c))P_{nb}(t-1, l_1) \tag{2.31}$$

$$P_b(t, l_1) \mathrel{+}= P_{ctc}(t, num(c))P_b(t-1, l_1) \tag{2.32}$$

После вычисления вероятностей всех префиксов на шаге t, префиксы сортируются по уменьшению суммарной вероятности $P_b(t, l) + P_{nb}(t, l)$. После этого

выбирают k лучших префиксов. Число k называется шириной луча в алгоритме лучевого поиска. Множество лучших префиксов на шаге t обозначим как R_t.

После этого итерация алгоритма повторяется. При достижении конечного шага времени T из множества R_T выбирается самый вероятный префикс - он и является результатом декодирования префиксным поиском. Пример реализации на языке Python можно увидеть в листинге 5.1. На рисунке 2.5 проиллюстрированы первые шаги алгоритма префиксного поиска.

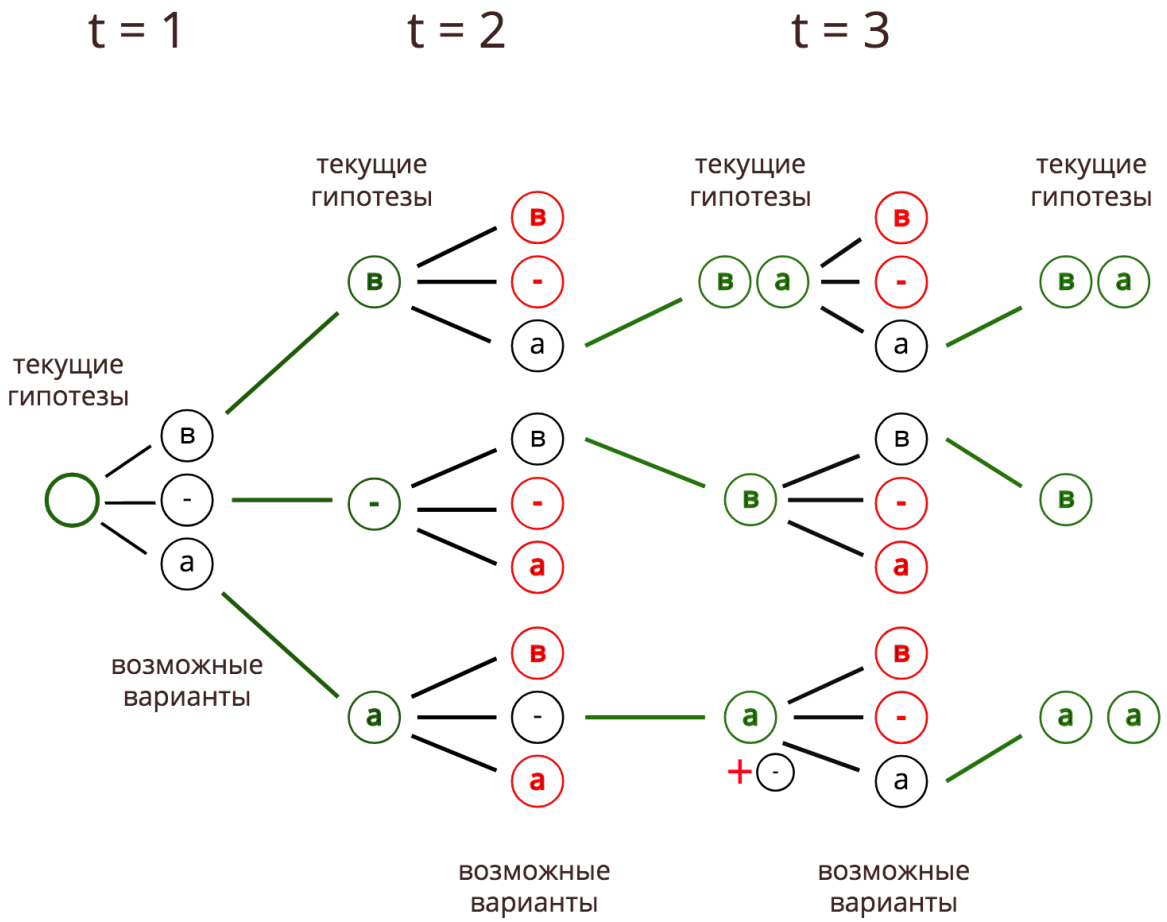


Рис. 2.5 — Первые шаги алгоритма префиксного поиска с размером расширенного алфавита 3 и шириной луча 3. Символом «-» обозначен символ *пропуска*.

2.6 Языковая модель

В архитектуре DeepSpeech для повышения точности распознавания речи используется вероятностная языковая модель. Она представляет собой базу данных, в которой хранятся оценки вероятностей встречи последовательностей

слов в языке. Каждой последовательности слов длины от 1 до N сопоставлено число, характеризующее ее вероятность. Последовательности, состоящие из n слов называют n-граммами. Максимальное число N длины последовательностей слов определяет размерность модели. Если требуется оценить последовательность слов, длина которой больше размерности модели или оцениваемая последовательность не найдена в базе, то для оценки используются вероятности последовательностей меньшего размера вплоть до 1-грамм. В реализации Mozilla DeepSpeech для создания и осуществления запросов к языковой модели используется инструментарий KenLM [28]. Принцип построения вероятностной языковой модели описан в работах [28; 44].

2.6.1 Использование языковой модели при декодировании CTC матрицы

Языковая модель используется в формуле 2.29 в алгоритме лучевого поиска, описанного в части 2.5.

Значение $P_{LM}(l_1)$ оценивает вероятность встретить в речи последовательность слов l_1. Языковая модель увеличивает вероятность тех лучей, которые состоят из последовательностей слов, чаще встречающихся в языке, и, соответственно, уменьшает вероятность лучей с неправдоподобными последовательностями или ошибками в словах.

2.7 Выводы

– Описана структура, используемой в качестве акустической модели, глубокой нейронной сети.

– Описан процесс обучения нейронной сети.

– Выведена формула функции потерь CTC Loss, а также формула ее частной производной, используемая для обучения нейронной сети методом стохастического градиентного спуска.

– Пошагово описан алгоритм лучевого поиска для декодирования матрицы предсказания нейронной сети.

Глава 3
Сбор данных для обучения акустической модели

Для обучения глубоких нейронных сетей распознаванию речи требуются сотни часов тренировочных данных. Из-за отсутствия в открытом доступе достаточно больших наборов данных для обучения распознаванию русской речи, наиболее трудозатратной задачей настоящей работы стала задача сбора данных. Ввиду ограниченности ресурсов и времени, оптимальным решением явилась разработка методов для автоматического сбора и фильтрации данных из открытых источников. В этой главе представлены несколько методов автоматического создания набора данных для обучения, рассмотрены достоинства и недостатки каждого из них.

3.1 Формат набора данных

Набор данных представляет собой совокупность из 3-х текстовых файлов формата csv: train.csv, dev.csv и test.csv и соответствующих директорий с аудиофайлами отрывков речи в формате wav. Аудиофайлы формата wav имеют единственный звуковой канал (моно), частоту дискретизации 16000 Гц и закодированы с шириной 2 байта для каждого значения.

Также из алгоритма декодирования CTC матрицы вытекает ограничение на соотношение длины аудио и количество символов в транскрипте, а именно: количество шагов в матрице CTC должно быть больше, чем количество символов в транскрипте.

Каждый из csv файлов содержит таблицу данных со столбцами *wav_filename*, *wav_filesize* и *transcript*. Столбец *wav_filename* указывает полный путь до файла wav, а в столбце *transcript* указан текст, произносимый в этом аудио-файле. В столбце *wav_filesize* указывается размер отрезка аудио в байтах. Информация о размере файла используется для сортировки примеров при обучении нейронной сети. Пример формата csv файла можно увидеть ниже в таблице 3.1

Таблица 3.1

Пример формата данных для сопоставления аудио и транскрипта.

wav_filename	wav_filesize	transcript
/tmp/vox/ru_0081.wav	220044	мышь понемногу ожила подняла нос стала шевелить усами умылась
/tmp/vox/ru_0082.wav	280044	покалывало грудь стучала кровь в виски но дышалось легко воздух был тонок и сух
/tmp/vox/ru_0084.wav	244044	идти было необычайно легко хотя ноги и вязли по щиколотку в рассыпающей почве
/tmp/vox/ru_0085.wav	252044	из под ног выбегали животные похожие на каменных ящериц ярко оранжевые с зубчатым хребтом
и.т.д.		

3.2 Создание корпуса речи с использованием аудио из видеофайлов сервиса YouTube, а также сопоставленных пользовательских и автоматически-сгенерированных субтитров

В начале процесса получения транскрибированных аудио-файлов производится поиск видео при помощи YouTube API. Устанавливается флаг для поиска видео только с субтитрами, которые были добавлены автором видео. Для поиска по множеству запросов в цикле существует файл со списком текстовых запросов. Этот файл наполняется запросами по определенной теме вручную. Для получения различных запросов относящихся к как-либо теме был использован сервис Yandex WordStat и расширение Yandex Wordstat Assistant. После поиска видео по запросам, найденные идентификаторы видео сохраняются в список для последующей обработки.

Далее для каждого из видео из полученного списка производится попытка скачивания автоматически распознанных субтитров YouTube в формате субтитров vtt. Для скачивания субтитров и аудио с YouTube была использована утилита youtube-dl. В автоматических субтитрах отмечено время начала каждого из распознанных слов. В python скрипте производится построение соответствия слова и его временного промежутка на основе этих данных.

После этого скачиваются загруженные автором субтитры (они есть, потому что видео найдено с помощью соответствующего фильтра). Далее для каждого размеченного в авторских субтитрах отрезка производится поиск точ-

ного соответствия (по словам) в автоматически сгенерированных субтитрах. Такого соответствия может не найтись, например, потому что какие-то слова были не распознаны или потому что авторские субтитры не соответствуют речи в видео. Если же соответствие найдено, то мы получаем более точные, чем в авторских субтитрах, отметки времени начала и конца речи, а также уверенность в том, что в аудио файле присутствует определенный текст (так как автоматическая система распознавания “услышала” данный текст в аудио). Получить точные отметки времени начала и конца речи важно, чтобы не обрезать речь на полуслове и не включить речь, которой нет в транскрипте. Если хоть одно соответствие было найдено то производится загрузка аудио-файла и его разбиение.

Так как отметки времени, полученные из файла автоматически сгенерированных субтитров все равно иногда оказываются посередине произносимого слова, то производится коррекция этих отметок до ближайшего места тишины в пределах 0.5 секунды. Для определения тишины в аудио используется Python-оболочка утилиты Voice Activity Detector из open-source проекта WebRTC.

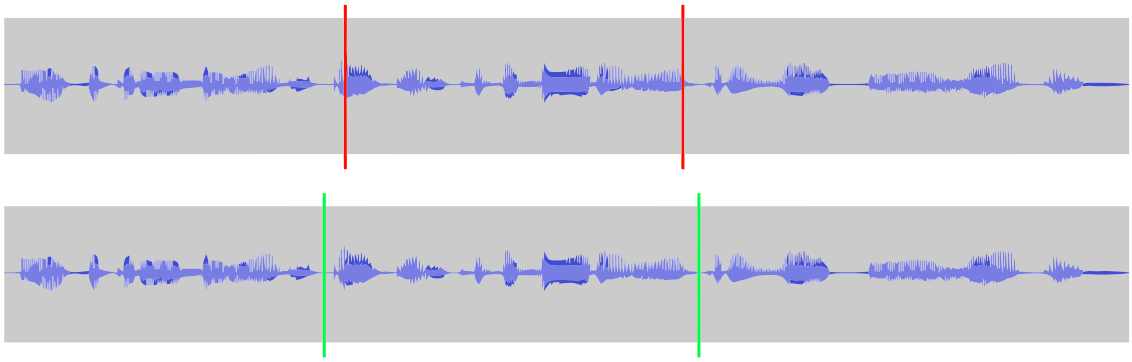


Рис. 3.1 — Коррекция обрезки отрывков с помощью детектора речи

Для каждого видео в отдельной директории хранятся все данные для этого видео: исходный аудио-файл, конвертированный wav аудио-файл, *parts.csv* - документ с транскриптами для успешно обработанных отрезков этого видео, директория *parts* с wav аудио-отрезками, а также файл *stats.csv*, где сохранена статистика об обработке этого видео: количество секунд речи и количество успешно извлеченных аудио-отрезков.

Для успешно скорректированного отрезка речи (если коррекция была нужна), из аудио файла вырезается размеченный отрезок аудио в формате wav и сохраняется в директорию parts, а в файл *parts.csv* записывается полный путь до этого отрезка, размер отрезка аудио в байтах и соответствующий ему

текст. Если на этапе обработки видео происходит какая-то ошибка, то это видео добавляется в список проблемных видео с описанием проблемы, чтобы в дальнейшем не тратить на него ресурсы в случае если оно опять попадается в поиске по какому-либо запросу. Идентификаторы успешно обработанных видео также сохраняются в отдельный файл, для предотвращения повторной обработки и статистики.

Данный подход оказался самым точным в смысле соответствия текста транскрипта и произнесенного в аудио текста. Недостатком данного подхода оказалась слишком высокая избирательность алгоритма: если для строки авторских субтитров алгоритм распознавания речи YouTube распознал хотя-бы одно слово неверно, то эта строка выбрасывается из набора данных.

Вследствие этого скорость сборки корпуса речи оказалась очень низкой - около 2 часов речи за один час реального времени на указанном оборудовании и скорости интернет-соединения. Также по статистике сбора данных было понятно, что только около 2.5% от всего скачанного аудио входит в результирующий корпус. В итоге, таким способом за несколько дней удалось создать корпус объемом 140 часов речи.

Исходя из того, что во многих источниках указывается необходимый объем корпуса для обучения глубокой нейронной сети распознаванию речи равный 400-1000 часам, данный подход виделся не оптимальным в условиях ограниченного времени и количества видео с субтитрами в русском сегменте YouTube.

3.3 Создание корпуса речи из аудио YouTube, разбиением по промежуткам тишины и последующим подбором слов под разбитые отрезки

В начале процесса также получается каким-либо путем список идентификаторов видео YouTube. Далее отфильтровываются все видео, для которых не были созданы автоматические субтитры YouTube. Для видео с автоматическими субтитрами производится скачивание аудио и файла vtt автосубтитров.

Аудио конвертируется в формат wav с параметрами: 16кГц, моно, 16 бит. Из скачанного файла субтитров vtt извлекается распознанное системой транс-

крибирования YouTube примерное (с ошибкой около 0.3-0.5 секунд) время начала и конца каждого слова. Аудио разбивается в промежутках между речью (которые определяются с помощью детектора речи WebRTC VAD) на отрезки от 5 до 20 секунд.

После этого для каждого из отрезков выбираются слова, по времени принадлежащие ему. Выбранные слова сортируются по времени начала их произнесения и собираются в одну строку, разделенную символом пробела. После этого к аудио применяется корректировка громкости чтобы довести его до уровня -10 децибел и полосовой частотный фильтр (bandpass filter) с границами 200Гц и 3000Гц.

Данный способ генерации корпуса речи намного быстрее предыдущего по двум причинам.

Первая это то, что точная обрезка аудио по промежуткам тишины производится сразу на первом этапе последовательно для всего аудиофайла, в отличие от предыдущего способа, где для каждого отрезка производилась попытка скорректировать начало и конец аудио.

А вторая состоит в том, что в этом способе не используются пользовательские субтитры, что дает доступ к гораздо большему объему видео ценой качества транскрипта.

Получение 100 часов речи занимает 1 час реального времени. Что в сравнении с предыдущим способом в 50 раз быстрее.

3.4 Создание корпуса речи из записей и транскриптов передач «Эхо Москвы»

Еще одним источником материала для обучения стал сайт радио станции «Эхо Москвы», где в открытом доступе предоставлены аудиофайлы записей прередач и сответствующие транскрипты. На сайте доступно по меньшей мере две тысячи записей, что делает данный ресурс привлекательным источником для создания достаточно большого корпуса речи. Каждая передача представляет собой запись длиной около 40 минут.

В процессе исследования возможности извлечения полезных данных обнаружилось несколько проблем:

1. Текст транскрипта никак не прикреплен к аудио записи, поэтому непонятно, как разделить аудио на небольшие отрезки и сопоставить им текст чтобы создать примеры для обучения.
2. В записях передач встречаются заставки и реклама, в которых присутствует музыка и речь, которые никак не отражены в текстовой версии передачи.
3. Так как все передачи представляют собой диалоги, то очень часто встречаются моменты в записи, когда оба собеседника говорят в одно и то же время. Отфильтровывание таких фрагментов также представляется сложной задачей.
4. В диалоговой речи часто встречаются звуки обозначающие процесс мышления собеседников и они также не включены в текстовую версию. Такие звуки из аудио необходимо исключать. Также довольно часто в речи встречается повторение одного слова последовательно несколько раз.

Первые две проблемы удалось решить, о чем будет написано далее. Последние же две остались нерешенными.

3.4.1 Автоматическое выравнивание текста по аудио при помощи утилиты aeneas

Существует класс программ для автоматического выравнивания отрывков текста по аудио. Эти утилиты используются, например, для автоматической синхронизации текстового и аудио варианта книг.

В основном для этого применяют 2 подхода:

– использование систем распознавания речи для определения границ текста в аудио
– использование систем для синтеза речи (Text to Speech, TTS) и алгоритма автоматической трансформации временной шкалы (Dynamic Time Wrapping, DTW) для сравнения и сопоставления двух аудио [37]

Из-за отсутствия надежной системы распознавания русской речи в открытом доступе, для выравнивания был выбран второй подход. Выравнивание производилось с помощью утилиты aeneas. Но сразу возникла проблема с рекламой, которой не было в транскрипте. При этом подходе необходимо было вырезать рекламные блоки и заставки из аудио.

3.4.2 Автоматическое распознавание и вырезание рекламных блоков и заставок из радио передач

В рассматриваемых радио передачах было 2 типа аудио, которые необходимо было вырезать: заставки и рекламные блоки.

Для заставки в радио передачах использовалось одно и то же аудио, поэтому его получилось вырезать с помощью метода поиска звукового отпечатка по заготовленной заранее базе данных [27].Была использована Python библиотека *audfprint*, позволяющая создавать базу данных отпечатков, а также искать в аудио-файле известные записи. База данных содержала лишь одну запись заставки. Данный метод позволил определять начало и конец заставки с точностью до 1-2 секунд, даже если в записи радио-передачи была включена только минимальная часть заставки.

Для вырезания рекламных блоков описанный выше метод не подходил, так как реклама всегда начиналась по-разному.

Было замечено, что частота речи и громкость в рекламных блоках имеет более высокие значения, чем в остальной передаче (см. рисунки 3.2 и 3.3). С помощью анализа амплитуд аудио-сигнала были выявлены участки с высокой плотностью речи и громкостью, которые почти на всех записях соответствовали рекламе и заставкам с погрешностью в 5-10 секунд. Пример выделения рекламы в аудио файле можно увидеть на рисунке 3.4.

Заставка в передаче могла звучать в самом начале, а также сразу после рекламы, обозначая ее конец. Поэтому при определении времени конца рекламного блока был использован более точный метод поиска по аудио-отпечатку.

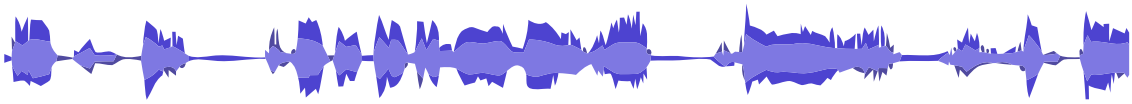


Рис. 3.2 — Пример аудио-сигнала диалога

Рис. 3.3 — Пример аудио-сигнала, в котором звучит реклама

Таким образом удалось вырезать рекламные блоки и заставки из радио-передач и, тем самым, подготовить аудио-файл для работы автоматического DTW выравнивателя.

Но из-за наличия последних двух проблем с записями аудио-передач, включающих дополнительные звуки и перекрывающуюся речь, автоматический выравниватель оказывался рассинхронизирован на большей части аудио, что помешало использовать этот метод в автоматическом создании корпуса речи.

3.5 Наилучший из рассмотренных метод автоматического создания корпуса речи

На основе проведенных исследований, самым оптимальным по затратам ресурсов и качеству оказался способ (3.3), при котором сначала весь аудио-файл разбивается на части с помощью детектора тишины, а уже потом к нарезанным кускам подбираются слова из автоматических субтитров YouTube.

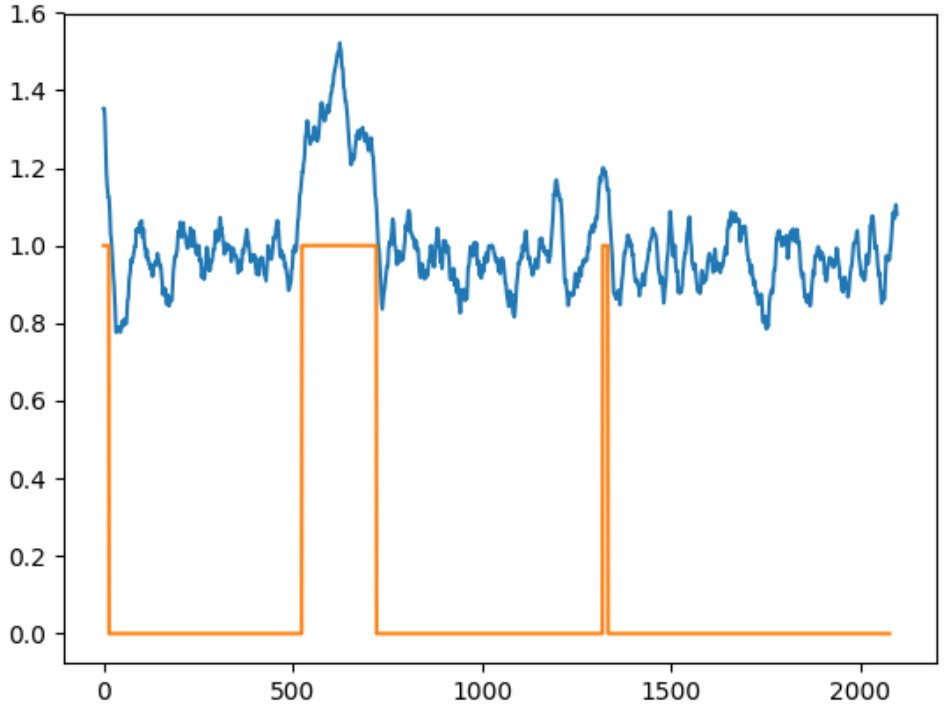


Рис. 3.4 — Успешно выделенные промежутки рекламы и заставок в записи передачи. Синим цветом обозначен уровень плотности отрезков аудио с высокой средней амплитудой. Оранжевым цветом на отрезках со значением 1 обозначены промежутки, классифицированные как реклама.

3.6 Автоматическая очистка корпуса речи от неверно-обрезанных примеров с помощью промежуточной акустической модели

Несмотря на то, что (3.3) является лучшим из рассмотренных автоматических методов, качество набора данных, получаемого этим способом все еще несравнимо с качеством наборов данных, подготовленных человеком.

Источником неточностей является определение границ слов в аудио. В ходе оценки 1000-часового набора данных «yt-vad-1k» полученного методом (3.3), было установлено, что более 30% примеров набора имеет лишние или недостающие слова в начале или конце транскрипта.

Для решения этой проблемы была использована промежуточная акустическая модель, обученная на 1000-часовом наборе «yt-vad-1k» с ошибками обрезки. Модель имела ошибку распознавания в 35.2% WER и 11% CER на те-

стовом поднаборе «yt-vad-1k-test». Для каждого из примеров было проведено предсказание транскрипта. После чего ожидаемый и предсказанный транскрипты были сравнены по 15 первым и 15 последним символам с помощью относительного расстояния Левенштейна. Если значение относительного расстояния было больше 0.5 в начале или конце, то пример исключался из набора данных. Таким образом удалось автоматически очистить наборы данных от примеров с неверной обрезкой и получить два новых набора: «voxforge-ru-clean» и «yt-vad-650-clean» объемом 11.5 и 650 часов соответственно.

3.7 Выводы

– Рассмотрены несколько способов автоматического сбора данных из различных источников для обучения нейронной сети распознаванию речи.

– Выбран оптимальный способ автоматического сбора данных для обучения, позволяющий за короткое время собирать сотни часов транскрибированной речи с видеохостига YouTube.

– Выбранным методом собран набор данных «yt-vad-1k» состоящий из 1000 часов транскрибированной речи, включающий в себя речь тысяч людей с различными условиями звукозаписи и фоновыми шумами.

– Предложен и применен способ автоматической очистки данных от неверно обрезанных примеров. Данным способом получены еще два набора данных: «voxforge-ru-clean» и «yt-vad-650-clean».

Глава 4
Обучение модели

Имея архитектуру нейронной сети, функцию потерь и данные для обучения, перейдем к тренировке акустической модели.

4.1 Контейнеризация процесса обучения с помощью технологии Docker

Для упрощения переносимости кода и необходимого для обучения программного окружения на другие ЭВМ, процесс обучения был контейнеризован с помощью технологии Docker. Данная технология позволяет упаковать все необходимые программные компоненты в единственный файл образа контейнера. После чего, данный образ можно автоматически развернуть в контейнеры на множестве других ЭВМ без временных затрат и человеческих ошибок при настройке программного окружения.

Для создания образа с помощью технологии Docker используется текстовое описание в файле «Dockerfile», определяющее программное окружение контейнера. Описание контейнера состоит из базового образа (чаще всего это образ операционной системы) и набора команд, настраивающего программное окружение в контейнере.

Образ контейнера, использованный для обучения в настоящей работе состоял из базового образа ОС Ubuntu 16.04 с уже включенной поддержкой оптимизаций для графических процессоров Nvidia. Также в Dockerfile была описана сборка фреймворка TensorFlow с поддержкой GPU и оптимизациями под используемый тип процессора (1.1), а также сборка проекта DeepSpeech. Подробное описание всех команд можно найти в листинге Dockerfile 5.2.

Для доступа к графическим процессорам Nvidia, позволяющим значительно ускорить обучение была использована утилита nvidia-docker.

После успешной сборки образа с помощью Dockerfile, на основе образа был создан контейнер, в котором производились все дальнейшие эксперименты по обучению.

4.2 Обзор гиперпараметров обучения

Для настройки процесса обучения доступны следующие основные гиперпараметры:

1. **n_hidden** - число нейронов в скрытых слоях нейронной сети. Считается [29; 50], что данный параметр должен быть выбран в зависимости от объема обучающих данных и его разнообразия. В обучении модели для английского языка в проекте Mozilla DeepSpeech стандартное значение равно 2048. Максимальное значение ограничено объемом доступной видеопамяти.
2. **learning_rate** - скорость обучения. Стандартное значение из проекта Mozilla DeepSpeech: 0.0001.
3. **epoch** - число эпох, или число полных прохождений процесса обучения по всем тренировочным данным.
4. **train_batch_size, dev_batch_size, test_batch_size** - количество примеров в одном батче для тренировки, валидации и тестирования соответственно. Выбирается максимальное значения исходя из объема видеопамяти. Обычно используют значения от 8 до 64.
5. **dropout_rate** - вероятность исключения нейрона из вычислений во время шага обучения. Считается [17], что данный подход позволяет избежать переобучения сети. Стандартное значение: 0.3.

Также очень важным параметром, не включенным в список являются данные для обучения и тестирования. Именно от их качества зависит конечный результат.

4.3 Описание наборов данных, используемых в экспериментах

Эксперименты проводились с использованием трех наборов данных: «yt-vad-1k», «yt-vad-650-clean» и «voxforge-ru-clean». Наборы данных состоят из 1000, 650 и 11.5 часов русской речи соответственно. Наборы «yt-vad-1k» и «yt-vad-650-clean» состоят из речи тысяч человек в видео-роликах YouTube, записанных в различных шумовых условиях. Набор «voxforge-ru-clean» состоит из более чистых записей чтения вслух отрывков литературных произведений.

4.4 Структура экспериментов

Обучение нейронной сети в большинстве экспериментов проводилось до того момента, когда на протяжении нескольких эпох метрика потерь CTC Loss перестает снижаться или начинает увеличиваться. В остальных экспериментах обучение проводилось до достижения минимального показателя WER.

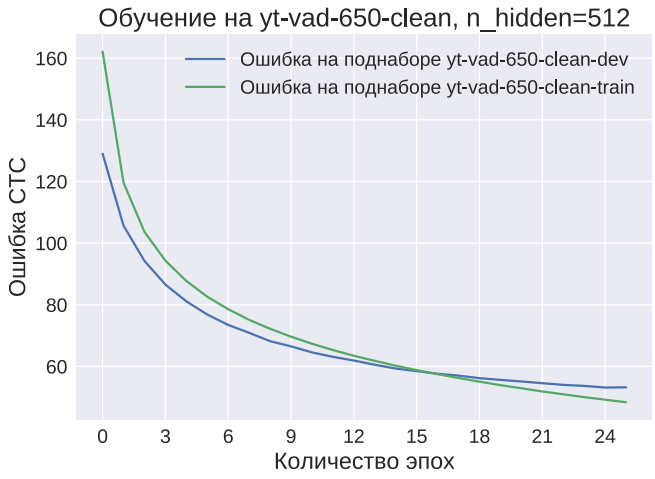
В начале были произведены эксперименты на основе части набора «yt-vad-650-clean» для определения оптимального количества нейронов в скрытых слоях нейронной сети (параметр **n_hidden**). При тестировании применялась языковая модель с длиной последовательности 3. Лучший вариант выбирался на основе минимального показателя WER.

После экспериментов по выявлению наилучшего значения параметра **n_hidden** были проведены эксперименты на полном объеме данных обоих наборов для определения минимально достижимой ошибки WER. Также был проведен эксперимент с дообучением модели, обученной на корпусе «yt-vad-1k», на данных «yt-vad-650-clean». Тестирование проводилось на наборах данных «yt-vad-650-clean-test» и «voxforge-ru-clean-test». Для каждого набора были проведены тестирования с использованием языковой модели и без. При тестировании с языковой моделью применялась языковая модель с длиной последовательности 3.

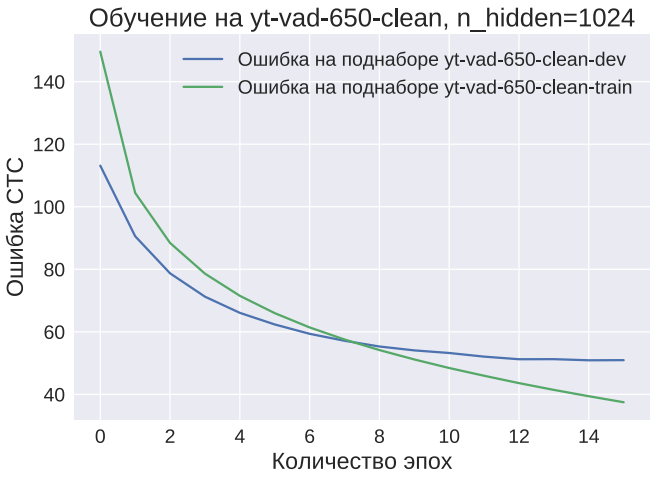
Далее были проведены эксперименты по выявлению оптимальной размерности языковой модели. Было произведено тестирование лучшей акустической модели из предыдущих экспериментов с разными конфигурациями языковой модели. Лучшее значение размерности выбиралось на основе минимального значения WER.

4.5 Эксперименты

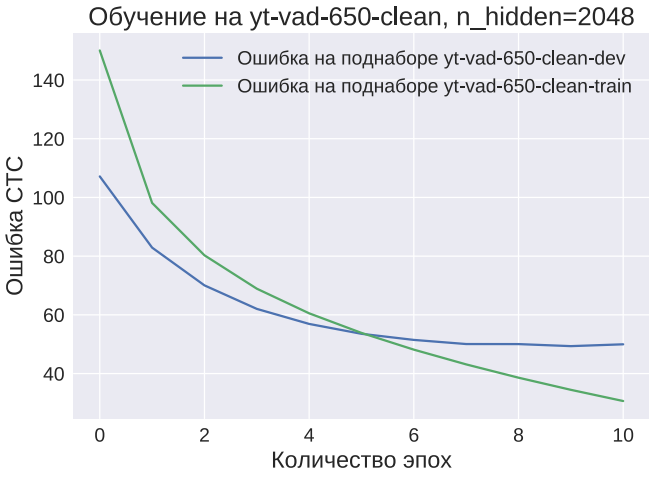
4.5.1 Оптимизация числа нейронов в скрытых слоях нейронной сети



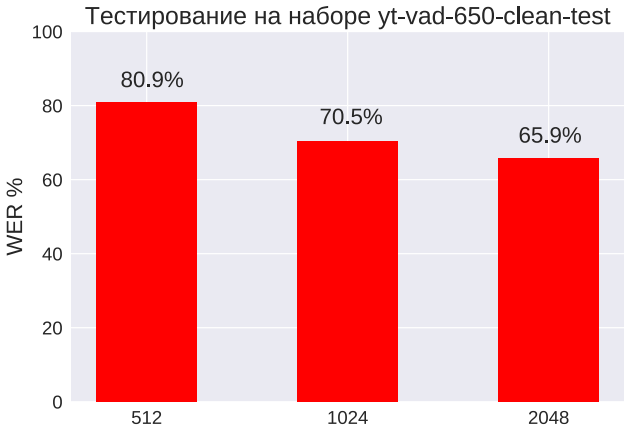
а) n_hidden=512



б) n_hidden=1024



в) n_hidden=2048



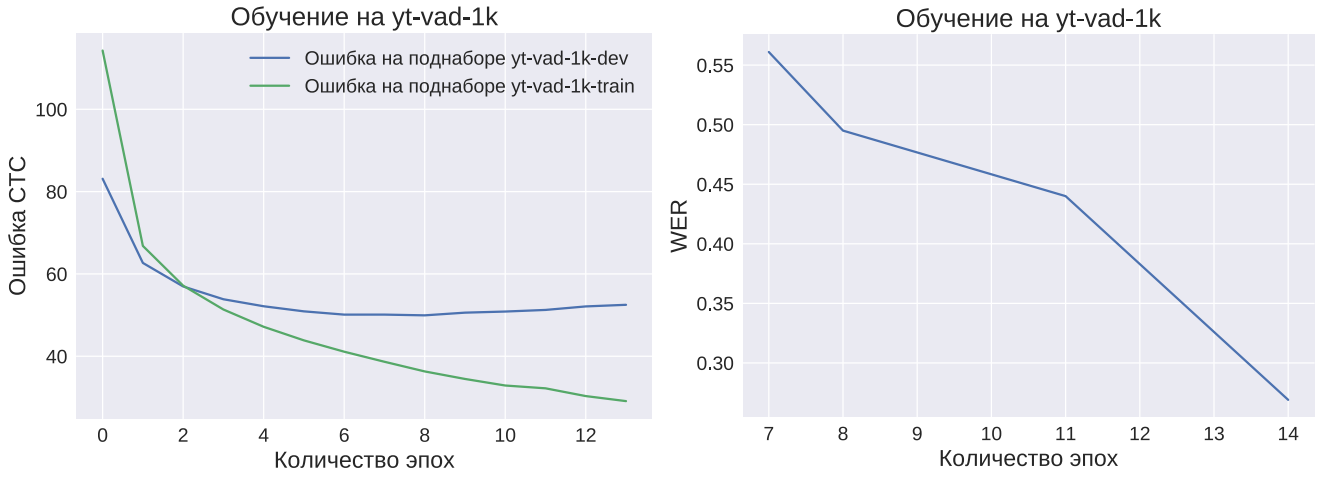
г) Сравнение по WER

Рис. 4.1

Из результатов экспериментов видно, что при увеличении количества нейронов в скрытых слоях нейронной сети от 512 до 2048 наблюдается снижение показателя WER. Дальнейшее увеличение количества нейронов привело к ошибкам, связанным с недостатком видеопамяти в текущей конфигурации 1.1 на некоторых аудио-файлах (уже при значении n_nidden=2500), поэтому опти-

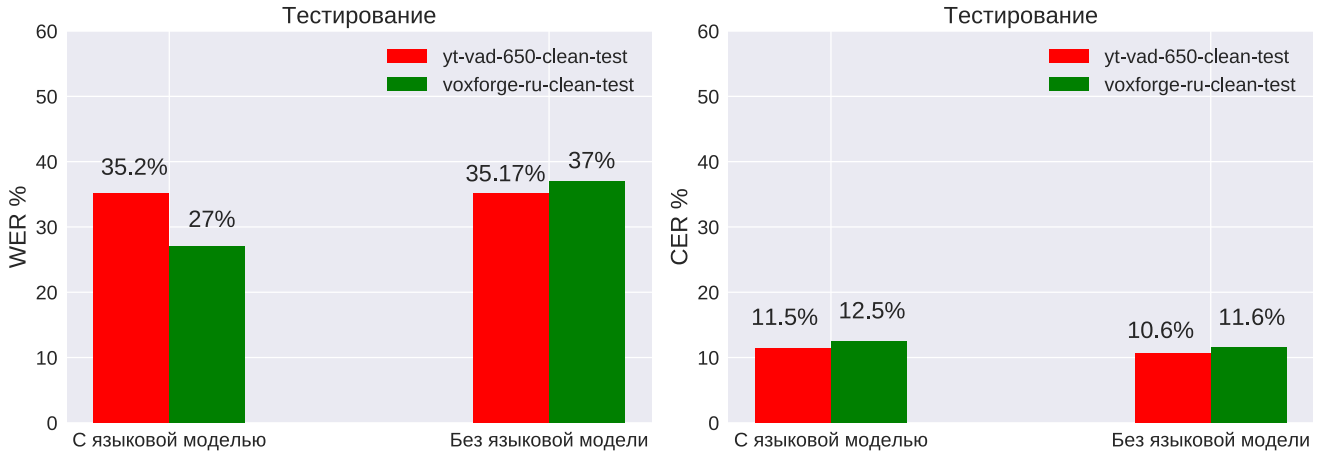
мальным выбрано значение равное 2048. Данное значение будет применяться в последующих экспериментах.

4.5.2 Обучение на всем наборе «yt-vad-1k-train»



а) Ошибка CTC б) WER

Рис. 4.2 — Процесс обучения модели на корпусе «yt-vad-1k».



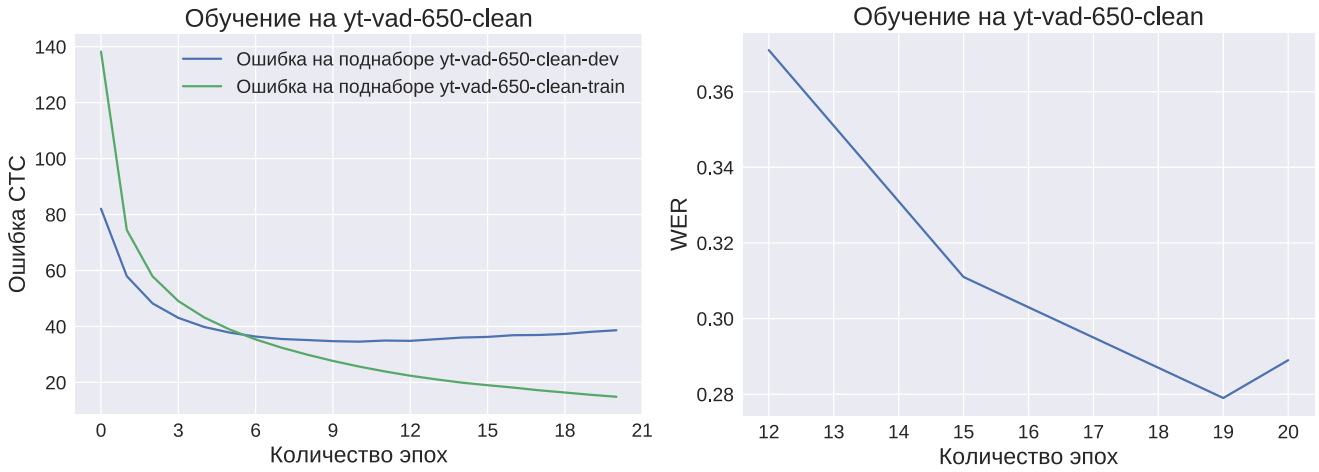
а) WER б) CER

Рис. 4.3 — Результаты тестирования модели, обученной на наборе «yt-vad-1k-train».

В результате обучения на наборе «yt-vad-1k-train» достигнуто минимальные значения ошибок 27% WER и 10.6% CER. Получено десятипроцентное улучшение WER на наборе «voxforge-ru-clean-test» при использовании языковой модели. Замечено незначительное увеличение показателя CER при использовании языковой модели на обоих наборах данных. Несмотря на то, что исходя из показателя ошибки CTC Loss переобучение началось на второй эпохе, показатель WER продолжал снижаться до 14 эпохи, даже при увеличении значение

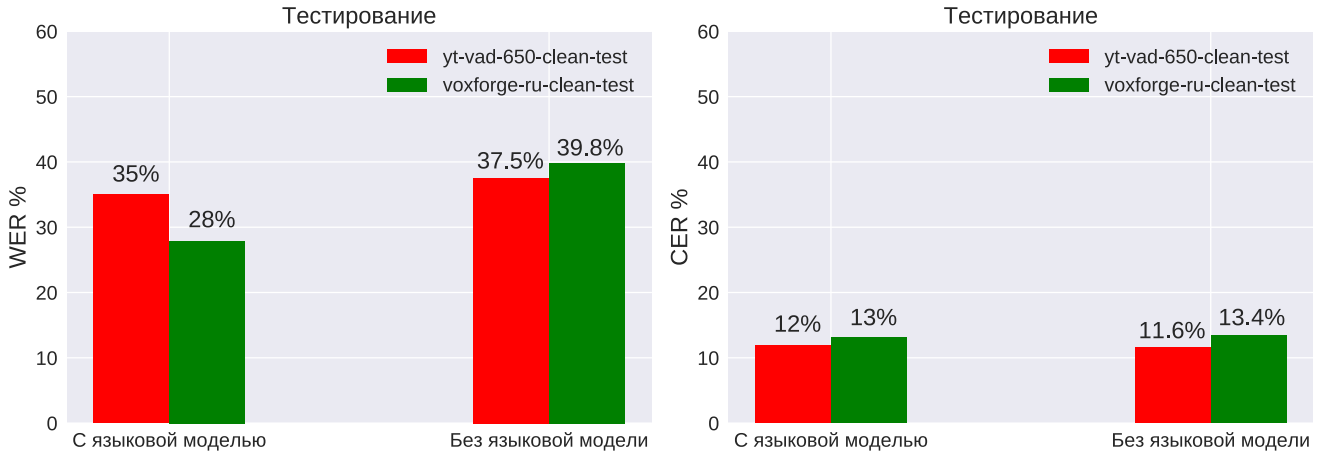
CTC Loss, что можно объяснить различием в способе подсчета CTC Loss и WER.

4.5.3 Обучение на всем наборе «yt-vad-650-clean-train»



а) Ошибка CTC б) WER

Рис. 4.4 — Процесс обучения модели на корпусе «yt-vad-650-clean».



а) WER б) CER

Рис. 4.5 — Результаты тестирования модели, обученной на наборе «yt-vad-650-clean-train».

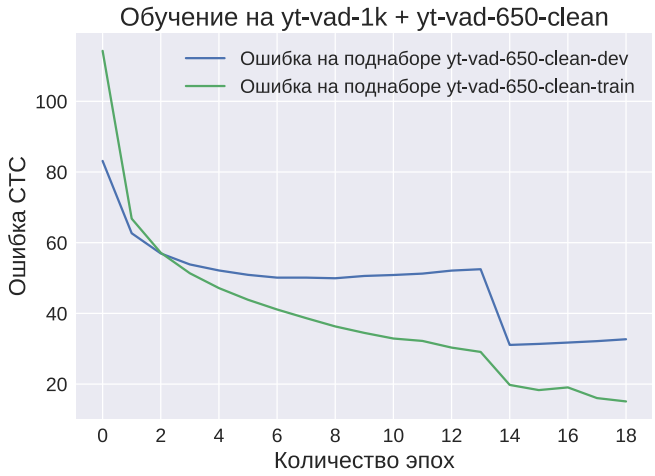
Из кривых обучения видно, что переобучение (если судить по показателю CTC Loss) в данном эксперименте произошло позже (на 6 эпохе), чем в предыдущем (на 2 эпохе). Также заметно улучшились минимальные значения CTC Loss на проверочном наборе с 50 до 35.

Из результатов тестирования модели, обученной на более чистом (по сравнению с «yt-vad-1k») корпусе «yt-vad-650-clean» заметно небольшое ухудшение минимальных показателей WER и CER. Данный результат можно объяснить

меньшим объемом данных - 650 часов по сравнению с 1000 часов в предыдущем эксперименте.

Как и в предыдущем эксперименте, тестирование с применением языковой модели дает лучшие показатели WER, тогда как показатели CER меняются незначительно.

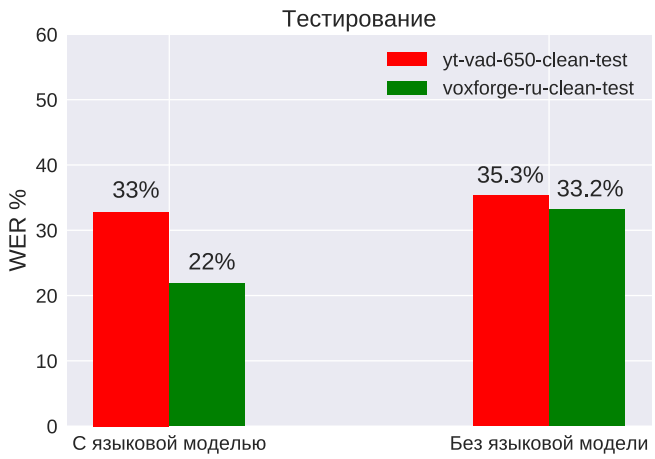
4.5.4 Дообучение модели, обученной на данных «yt-vad-1k-train», с помощью набора «yt-vad-650-clean-train»



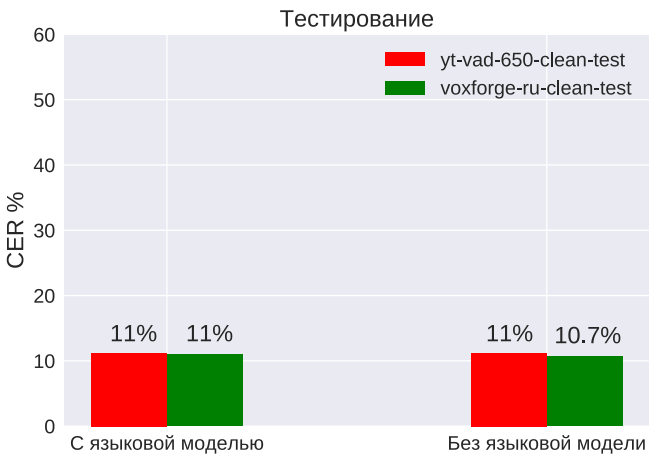
а) Ошибка CTC

б) WER

Рис. 4.6 — Процесс обучения модели на корпусе «yt-vad-1k» и «yt-vad-650-clean».



а) WER



б) CER

Рис. 4.7 — Результаты тестирования модели, обученной на наборе «yt-vad-1k-train» и «yt-vad-650-clean-train».

На графике кривых ошибки CTC четко виден момент смены корпуса «yt-vad-1k» на более чистый «yt-vad-650-clean» на 14-ой эпохе. Применяя дообучения модели, удалось снизить минимальный уровень WER с 27% до 22%.

4.5.5 Определение оптимальной размерности языковой модели

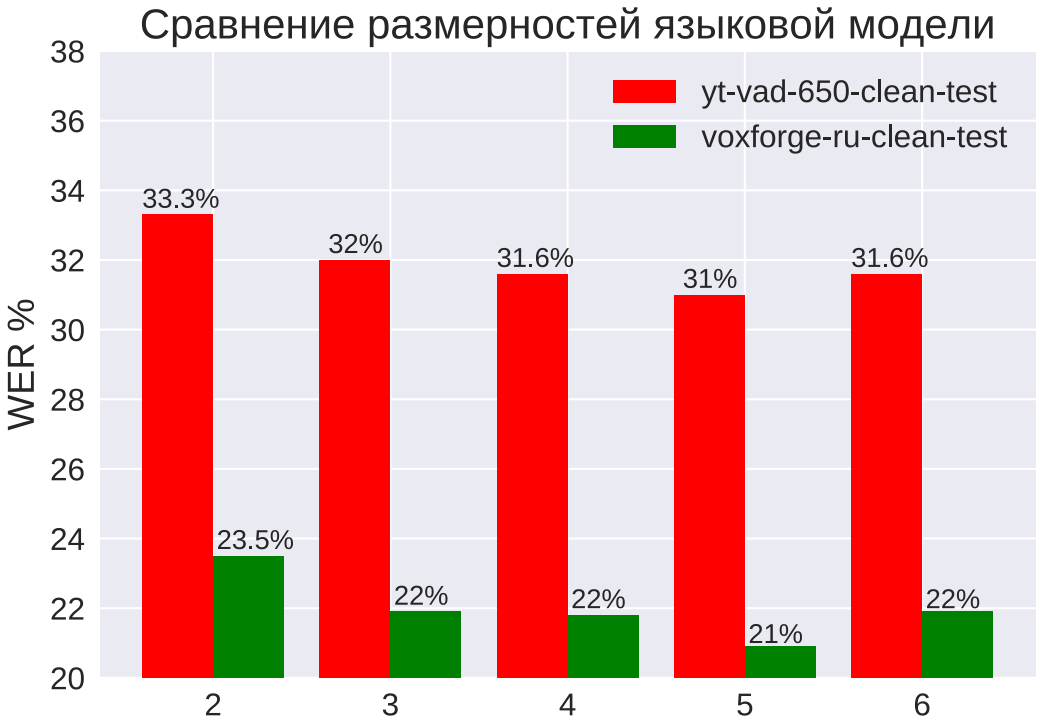


Рис. 4.8 — Сравнения конфигураций языковых моделей с различными длинами последовательностей слов на основе минимального уровня WER на наборах «yt-vad-650-clean-test» и «voxforge-ru-clean-test».

В данном эксперименте было рассмотрено влияние размерности языковой модели на точность распознавания. Из результатов эксперимента видно, что минимальный WER достигается при использовании языковой модели с максимальной длиной последовательности 5.

Кроме этого, в данном эксперимента были снижены минимальные значения WER: до 31% для более зашумленного набора и до 21% для набора с более чистой речью.

4.6 Выводы

– Определено оптимальное количество нейронов в скрытых слоях нейронной сети: 2048 на слой.

– Проведены эксперименты по обучению модели на двух корпусах, а также эксперимент с дообучением.

– Определена оптимальная размерность языковой модели: 5.

– На наборе данных незашумленной речи «voxforge-ru-clean-test» достигнут минимальный уровень ошибки в 21% WER.

– На наборе «yt-vad-650-clean-test» зашумленной речи достигнут минимальный уровень ошибки в 31% WER.

Глава 5
Создание системы индексации и поиска по речи, распознанной в медиафайлах

5.1 Описание системы в целом

Одним из возможных применений построенной системы распознавания речи является поиск по тексту произносимому в большом числе медиафайлов. Такая система поиска по текстовому запросу должна выдавать результат в виде списка медиафайлов и ссылок на конкретные отрезки времени в них, где произносится искомое слово или фраза.

В качестве примера была создана система индексации и поиска по видео веб-сайта YouTube. Данная система состоит из трех модулей: модуля распознавания речи в видео, модуля индексации распознанного текста и модуля поиска. Модуль поиска состоит из клиентской и серверной частей.

Рис. 5.1

5.2 Компоненты системы

5.2.1 Модуль распознавания

Модуль распознавания речи использует построенную ранее систему распознавания русской речи. Скаченный аудио файл для каждого видео разбивается на короткие отрезки длинной около 5 секунд. После чего для каждого из этих отрезков производится распознавание речи. Распознанный текст с соответствующим идентификатором видео и временными границами отрезка записывается в базу данных.

5.2.2 Модуль индексации

Модуль индексации производит подготовку индекса полнотекстового поиска. Каждое слово, заносимое в индекс обрабатывается с помощью алгоритма стемминга для нахождения основы слова, выражающей лексическое значение.

5.2.3 Модуль поиска

Модуль поиска при наличии готовой базы данных и индекса полнотекстового поиска производит поиск запрашиваемого текста по индексу, после чего для каждого соответствия выбирает элементы базы данных, где найденный текст сопоставлен с отрезком времени в конкретном видео.

Результаты поиска представляются пользователю на веб-странице в виде ссылок на сайт YouTube с отметками времени. Таким образом при переходе по ссылке проигрывается отрывок видео, где был произнесен запрашиваемый текст.

5.3 Реализация

Описанная система поиска была реализована на языках Python 2.7 и JavaScript.

В качестве базы данных была выбрана файловая БД SQLite, так как она не требует установки серверной части, что позволяет избавиться от лишних манипуляций по настройке примера. Работа с базой данных была реализована с помощью Python-библиотеки SQL Alchemy, позволившей упростить взаимодействие с БД используя объектно-реляционные преобразования из Python объектов в SQL и обратно.

Для индексирования текста, стемминга и полнотекстового поиска была использована Python-библиотека Whoosh. Эта библиотека также не требует дополнительных зависимостей кроме Python, а также поддерживает русский язык.

Интерфейс пользователя был реализован в виде веб-приложения. Для серверной части был использован Python-фреймворк Flask, а на стороне клиента JavaScript-фреймворк Vue.js. Поисковые запросы от клиентской стороны к серверной стороне были реализованы на основе технологии AJAX, что дало возможность не перезагружать веб-страницу для получения результатов поиска.

Код реализации приведен в листингах 5.3, 5.4, 5.5 и 5.6.

5.4 Выводы

– Построенная система распознавания речи применена для автоматического транскрибирования базы медиафайлов.

– Для базы распознанного текста создан полнотекстовый поисковой индекс с фильтрованием стоп-слов и поддержкой стемминга.

– Создан веб-интерфейс для взаимодействия пользователя с приложением поиска по речи в коллекции медиафайлов.

Заключение

Итоги выполненной работы:

1. Рассмотрен принцип работы системы распознавания речи с применением глубоких нейронных сетей на основе архитектуры DeepSpeech.
2. Исследованы методы автоматического сбора данных для создания корпусов транскрибированной речи.
3. Созданы два корпуса русской транскрибированной речи «yt-vad-1k» и «yt-vad-650-clean» объемом 1000 и 650 часов, пригодные для обучения современных систем распознавания речи с применением глубоких нейронных сетей.
4. Создана система распознавания русской речи, показывающая результат в 31% WER на корпусе речи с разнообразием голосов и шумов (набор данных «yt-vad-650-clean-test») и 21% WER на наборе с более чистой речью чтения текстов («voxforge-ru-clean-test»).
5. Создана система поиска по речи в большой коллекции медиафайлов с возможностью мгновенного перехода к воспроизведению искомого фрагмента.

Возможные применения результатов работы:

На основе метода создания корпуса транскрибированной речи для русского языка, предложенного в данной работе, могут быть собраны корпуса и для других языков. Архитектура DeepSpeech также имеет минимальное количество привязок к определенному языку (только задание алфавита), что позволяет использовать процесс обучения, описанный в данной работе, при создании систем распознавания речи для других языков. Известны успешные применения архитектуры DeepSpeech как для латинских языков, таких как английский, французский, так и для языков с гораздо большим размером алфавита, таких как китайский [15].

Дальнейшие перспективы разработки темы:

1. Дальнейшее исследование методов автоматического сбора корпусов транскрибированной речи и поиск источников для получения более качественных наборов данных.
2. Исследование применения других архитектур глубокой нейронной сети для повышения точности распознавания. Например, сравнение производительности при применении ячеек LSTM и GRU или рассмотрение применения систем, основанных на механизме внимания вместо CTC.
3. Повышение скорости распознавания путем распараллеливания алгоритма лучевого поиска или переноса вычислений на графический ускоритель.
4. Исследование возможности применения архитектуры для распознавания на менее мощных ЭВМ (таких как мобильные устройства).

Список сокращений

ASR Automatic Speech Recognition
BiRNN Bidirectional Recurrent Neural Network
CER Character Error Rate
CPU Central Processing Unit
CTC Connectionist Temporal Classification
DNN Deep Neural Network
DTW Dynamic Time Wrapping
GPU Graphics Processing Unit
GRU Gated Recurrent Unit
HMM Hidden Markov Model
LM Language Model
LSTM Long Short Term Memory
MFCC Mel-Frequency Cepstral Coefficients
ReLU Rectifier Linear Unit
RNN Recurrent Neural Network
SQL Structured Query Language
TTS Text to Speech
VAD Voice Activity Detector
WER Word Error Rate
БД База Данных
ОЗУ Оперативное Запоминающее Устройство
ОС Операционная Система
ЦПУ Центральное Процессорное Устройство
ЭВМ Электронно-вычислительная Машина

Словарь терминов

dropout: Значение вероятности исключения каждого отдельного нейрона из вычислений при единичной обработке данных нейронной сетью.

аудио-отпечаток: Сжатое цифровое представление, полученное детерминированным алгоритмом из аудио-сигнала.

батч: Группа примеров из набора данных.

видео-хостинг: Интернет-сервис, позволяющий пользователям загружать, хранить и просматривать видео файлы.

мел-частотные кепстральные коэффициенты: Представление кратковременной спектрального портрета звука в виде коэффициентов, получаемых при помощи косинусного преобразования Фурье.

рекуррентная нейронная сеть: Вид нейронной сети, где связи между элементами образуют направленную последовательность.

статистическая языковая модель: Распределение вероятностей по последовательностям слов определенной длины, приближенно отражающее реальное распределение, присутствующее в языке.

транскрипт: Текстовое представление речи.

фонема: Минимальная смыслоразличительная единица языка, соответствующая звуку речи.

фреймворк: Набор программных библиотек (инструментов) облегчающий и ускоряющий разработку программного обеспечения.

функция активации: Функция, вычисляющая выходной сигнал искусственного нейрона.

Список литературы

1. *Беленко М.*, *Балакшин П.* Сравнительный анализ систем распознавания речи с открытым кодом // Международный научно-исследовательский журнал. — 2017. — 04 (58) Часть 4. — С. 13—18.
2. *Меденников И. П.* Методы, алгоритмы и программные средства распознавания русской телефонной спонтанной речи: дис. ... канд. / Меденников Иван Павлович. — 2016.
3. A Journey to <10% Word Error Rate. — 2017. — Accessed: 2018-04-18. https://hacks.mozilla.org/2017/11/a-journey-to-10-word-error-rate/.
4. A novel connectionist system for unconstrained handwriting recognition / A. Graves [и др.] // IEEE transactions on pattern analysis and machine intelligence. — 2009. — Т. 31, № 5. — С. 855—868.
5. Addressing the rare word problem in neural machine translation / M.-T. Luong [и др.] // arXiv preprint arXiv:1410.8206. — 2014.
6. Advances in joint CTC-attention based end-to-end speech recognition with a deep CNN encoder and RNN-LM / T. Hori [и др.] // arXiv preprint arXiv:1706.02737. — 2017.
7. Attention-based models for speech recognition / J. K. Chorowski [и др.] // Advances in neural information processing systems. — 2015. — С. 577—585.
8. *Awan A. A.*, *Subramoni H.*, *Panda D. K.* An In-depth Performance Characterization of CPU-and GPU-based DNN Training on Modern Architectures // Proceedings of the Machine Learning on HPC Environments. — ACM. 2017. — С. 8.
9. *Bahdanau D.*, *Cho K.*, *Bengio Y.* Neural machine translation by jointly learning to align and translate // arXiv preprint arXiv:1409.0473. — 2014.
10. Cold fusion: Training seq2seq models together with language models / A. Sriram [и др.] // arXiv preprint arXiv:1708.06426. — 2017.

11. *Collobert R.*, *Weston J.* A unified architecture for natural language processing: Deep neural networks with multitask learning // Proceedings of the 25th international conference on Machine learning. — ACM. 2008. — С. 160—167.

12. Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks / A. Graves [и др.] // Proceedings of the 23rd international conference on Machine learning. — ACM. 2006. — С. 369—376.

13. CTC Networks and Language Models: Prefix Beam Search Explained. — 2018. — Accessed: 2018-04-18. https://medium.com/corti-ai/ctc-networks-and-language-models-prefix-beam-search-explained-c11d1ee23306.

14. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups / G. Hinton [и др.] // IEEE Signal Processing Magazine. — 2012. — Т. 29, № 6. — С. 82—97.

15. Deep Speech 2: End-to-End Speech Recognition in English and Mandarin / D. Amodei [и др.] // arXiv preprint arXiv:1512.02595. — 2015.

16. Deep speech: Scaling up end-to-end speech recognition / A. Hannun [и др.] // arXiv preprint arXiv:1412.5567. — 2014.

17. Dropout: A simple way to prevent neural networks from overfitting / N. Srivastava [и др.] // The Journal of Machine Learning Research. — 2014. — Т. 15, № 1. — С. 1929—1958.

18. Empirical evaluation of gated recurrent neural networks on sequence modeling / J. Chung [и др.] // arXiv preprint arXiv:1412.3555. — 2014.

19. End-to-end attention-based large vocabulary speech recognition / D. Bahdanau [и др.] // Acoustics, Speech and Signal Processing (ICASSP), 2016 IEEE International Conference on. — IEEE. 2016. — С. 4945—4949.

20. English conversational telephone speech recognition by humans and machines / G. Saon [и др.] // arXiv preprint arXiv:1703.02136. — 2017.

21. Google voice search: faster and more accurate / H. Sak [и др.] // Google Research blog. — 2015. — URL: https://research.googleblog.com/2015/09/google-voice-search-faster-and-more.html.

22. Google's neural machine translation system: Bridging the gap between human and machine translation / Y. Wu [и др.] // arXiv preprint arXiv:1609.08144. — 2016.

23. *Graves A.* Generating sequences with recurrent neural networks // arXiv preprint arXiv:1308.0850. — 2013.

24. *Graves A.* Supervised sequence labelling // Supervised sequence labelling with recurrent neural networks. — Springer, 2012. — С. 52—73.

25. *Graves A., Jaitly N.* Towards end-to-end speech recognition with recurrent neural networks // International Conference on Machine Learning. — 2014. — С. 1764—1772.

26. *Graves A., Mohamed A.-r., Hinton G.* Speech recognition with deep recurrent neural networks // Acoustics, speech and signal processing (icassp), 2013 ieee international conference on. — IEEE. 2013. — С. 6645—6649.

27. *Haitsma J., Kalker T.* A highly robust audio fingerprinting system. // Ismir. Т. 2002. — 2002. — С. 107—115.

28. *Heafield K.* KenLM: Faster and smaller language model queries // Proceedings of the Sixth Workshop on Statistical Machine Translation. — Association for Computational Linguistics. 2011. — С. 187—197.

29. How many hidden units should I use? — 2014. — Accessed: 2018-05-20. http://www.faqs.org/faqs/ai-faq/neural-nets/part3/section-10.html.

30. *Hu H., Zahorian S. A.* Dimensionality reduction methods for HMM phonetic recognition // Acoustics Speech and Signal Processing (ICASSP), 2010 IEEE International Conference on. — IEEE. 2010. — С. 4854—4857.

31. Hybrid CTC/Attention Architecture for End-to-End Speech Recognition / S. Watanabe [и др.] // IEEE Journal of Selected Topics in Signal Processing. — 2017. — Т. 11, № 8. — С. 1240—1253.

32. *Kingma D. P., Ba J.* Adam: A method for stochastic optimization // arXiv preprint arXiv:1412.6980. — 2014.

33. *Kiros R., Salakhutdinov R., Zemel R. S.* Unifying visual-semantic embeddings with multimodal neural language models // arXiv preprint arXiv:1411.2539. — 2014.

34. *Krizhevsky A., Sutskever I., Hinton G. E.* Imagenet classification with deep convolutional neural networks // Advances in neural information processing systems. — 2012. — С. 1097—1105.

35. *Le Q. V.*, *Jaitly N.*, *Hinton G. E.* A simple way to initialize recurrent networks of rectified linear units // arXiv preprint arXiv:1504.00941. — 2015.

36. *Medsker L.*, *Jain L.* Recurrent neural networks // Design and Applications. — 2001. — Т. 5.

37. *Müller M.* Dynamic time warping // Information retrieval for music and motion. — 2007. — С. 69—84.

38. *Nallasamy U.* Adaptation techniques to improve ASR performance on accented speakers: дис. ... канд. / Nallasamy Udhyakumar. — Carnegie Mellon University, 2016.

39. On the properties of neural machine translation: Encoder-decoder approaches / K. Cho [и др.] // arXiv preprint arXiv:1409.1259. — 2014.

40. *Pascanu R.*, *Mikolov T.*, *Bengio Y.* Understanding the exploding gradient problem // CoRR, abs/1211.5063. — 2012.

41. *Rabiner L. R.* A tutorial on hidden Markov models and selected applications in speech recognition // Proceedings of the IEEE. — 1989. — Т. 77, № 2. — С. 257—286.

42. *Sak H.*, *Senior A.*, *Beaufays F.* Long short-term memory recurrent neural network architectures for large scale acoustic modeling // Fifteenth annual conference of the international speech communication association. — 2014.

43. Say "Privet" to Alice, Yandex's Intelligent Assistant. — 2017. — Accessed: 2018-05-4. https://yandex.com/company/blog/say-privet-to-alice-yandex-s-intelligent-assistant/.

44. Scalable modified Kneser-Ney language model estimation / K. Heafield [и др.] // Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers). Т. 2. — 2013. — С. 690—696.

45. Show and tell: A neural image caption generator / O. Vinyals [и др.] // Computer Vision and Pattern Recognition (CVPR), 2015 IEEE Conference on. — IEEE. 2015. — С. 3156—3164.

46. Show, attend and tell: Neural image caption generation with visual attention / K. Xu [и др.] // International Conference on Machine Learning. — 2015. — С. 2048—2057.

47. *Simonyan K.*, *Zisserman A.* Very deep convolutional networks for large-scale image recognition // arXiv preprint arXiv:1409.1556. — 2014.

48. State-of-the-art speech recognition with sequence-to-sequence models / C.-C. Chiu [и др.] // arXiv preprint arXiv:1712.01769. — 2017.

49. The Microsoft 2016 conversational speech recognition system / W. Xiong [и др.] // Acoustics, Speech and Signal Processing (ICASSP), 2017 IEEE International Conference on. — IEEE. 2017. — С. 5255—5259.

50. *Weigend A.* On overfitting and the effective number of hidden units // Proceedings of the 1993 connectionist models summer school. Т. 1. — 1994. — С. 335—342.

51. *Yan Z.-J.*, *Huo Q.*, *Xu J.* A scalable approach to using DNN-derived features in GMM-HMM based acoustic modeling for LVCSR. // Interspeech. — 2013. — С. 104—108.

52. *Zaremba W.*, *Sutskever I.* Learning to execute // arXiv preprint arXiv:1410.4615. — 2014.

53. *Hochreiter S.*, *Schmidhuber J.* Long short-term memory // Neural computation. — 1997. — Т. 9, № 8. — С. 1735—1780.

Приложения

5.5 Реализация алгоритма префиксного поиска

Листинг 5.1

Реализация алгоритма префиксного поиска на языке Python2.7 с использованием языковой модели

```
1  # *= coding: utf-8 *=
2  import re
3  import numpy as np
4
5  # На входе в алгоритм поступают:
6  # * P_ctc — матрица вероятностей размера (35, T)
7  # (35 строк матрицы соответствуют 33 буквам русского алфавита, про
       белу и символу пропуска)
8  # * P_lm функция, оценивающая вероятность последовательности слов
       в языке
9  # * alpha  — параметр влияния языковой модели на вероятность префи
       кса
10 # * k — ширина луча (количество лучших префиксов, отбираемых на ка
       ждом шаге t)
11 # * beta — параметр влияния количества слов в префиксе на вероятно
       сть префикса
12 # * prune — порог вероятности сивола в CTC матрицы, ниже которого
       сивол не рассматривается
13 def prefix_beam_search(P_ctc, P_lm=None, k=25, alpha=0.30, beta=5,
       prune=0.001):
14
15     # если языковая модель не используется то возвращаемая вероятн
           ость равна 1
16     P_lm = (lambda l: 1) if P_lm is None else P_lm
```

```
17
18      # функция W возвращает последовательность слов из последовател
            ьности символов l
19      W = lambda l: re.findall(r'\w+[\s|>]', l)
20
21      # зададим алфавит — строчные русские буквы и пробел
22      alphabet = "абвгдеёжзийклмнопрстуфхцчшщъыьэюя "
23
24      # расширим алфавит символом пропуска
25      alphabet_ext = list(ascii_lowercase) + ['—']
26
27      # так как изначально нумерация в P_ctc списке идет с нуля доба
            вим
28      # дополнительный элемент в начале, чтобы
29      # нумеровать t начиная с 1
30      F = P_ctc.shape[1]
31      P_ctc = np.vstack((np.zeros(F), P_ctc))
32
33      # определим количество шагов времени T в P_ctc матрице
34      T = P_ctc.shape[0]
35
36      # шаг 1: Инициализация
37      # зададим словари, которые будут хранить вероятности
38      # для каждого префикса
39      Pb, Pnb = defaultdict(Counter), defaultdict(Counter)
40
41      # инициализируем вероятности
42      Pb[0][''] = 1
43      Pnb[0][''] = 0
44
45      # задаем список префиксов, состоящий из одного элемента — пуст
            ой строки
46      R_prev = ['']
47
```

```
48      # шаг 2: перебор всех шагов времени и прореживание алфавита
49      for t in range(1, T):
50
51          # Прореживаем алфавит исходя из вероятностей P_ctc матрицы
                .
52          # Символы расширеного алфавита с вероятностями меньше поро
                га prune
53          # рассматриваться не будут.
54          # Это позволяет существенно снизить время вычисления,
55          # и влияет на результат несущественно (при prune = 0.001).
56          pruned_alphabet = [alphabet_ext[i] for i in np.where(P_ctc
                [t] > prune)[0]]
57
58          # Перебираем все префиксы выбранные как лучшие на предыдущ
                ем шаге.
59          # Количество префиксов <= ширене луча k.
60          for l in R_prev:
61
62              # для каждого символа из прореженного алфавита
63              for c in pruned_alphabet:
64                  # находим индекс символа в расширенном алфавиье
65                  c_ix = alphabet_ext.index(c)
66
67
68                  # шаг 3: "удлиннение" префикса с помощью символа "
                        пропуск"
69                  if c == '—':
70                      Pb[t][l] += P_ctc[t][—1] * (Pb[t — 1][l] + Pnb
                            [t — 1][l])
71
72
73                  # шаг 4: удлиннение префикса с помощью символа алф
                        авита alphabet
74                  else:
```

```
75                  l_plus = l + c
76
77                  # Если символ совпадает с последним символом т
                        екущего префикса.
78                  if len(l) > 0 and c == l[-1]:
79                      Pnb[t][l_plus] += P_ctc[t][c_ix] * Pb[t -
                            1][l]
80                      Pnb[t][l] += P_ctc[t][c_ix] * Pnb[t - 1][l
                            ]
81
82
83                  # Если до этого уже существуют законченные сло
                        ва
84                  # и символ является пробелом или дошли до конц
                        а времени T,
85                  # то значит мы закончили (не первое) слово и т
                        огда используем языковую модель
86                  # для оценки получившейся последовательности с
                        лов.
87                  # Для настройки влияния языковой модели исполь
                        зуется параметр alpha.
88                  elif len(l.replace(' ', '')) > 0 and c == ' ':
89                      lm_prob = P_lm(l_plus.strip(' >')) **
                            alpha
90                      Pnb[t][l_plus] += lm_prob * P_ctc[t][c_ix]
                            * (Pb[t - 1][l] + Pnb[t - 1][l])
91
92                  # Иначе подсчитываем вероятность префикса допо
                        лненого символом c
93                  else:
94                      Pnb[t][l_plus] += P_ctc[t][c_ix] * (Pb[t -
                            1][l] + Pnb[t - 1][l])
95
96
```

```
97                      # шаг 5: учитываем вероятности префиксов,
98                      # которые были отсеяны из предыдущих выбранных
                            R_prev из-за низкой вероятности.
99                      if l_plus not in R_prev:
100                         Pb[t][l_plus] += P_ctc[t][-1] * (Pb[t -
                                1][l_plus] + Pnb[t - 1][l_plus])
101                         Pnb[t][l_plus] += P_ctc[t][c_ix] * Pnb[t -
                                1][l_plus]
102
103         # шаг 6: Выбираем k самых лучших (вероятных) префиксов на
                шаге t
104         # Параметр beta используется для настройки влияния
105         # количества слов в префиксе на его вероятность.
106         A_next = Pb[t] + Pnb[t]
107         sorter = lambda l: A_next[l] * (len(W(l)) + 1) ** beta
108         R_prev = sorted(A_next, key=sorter, reverse=True)[:k]
109
110     # шаг 7: возвращаем самый вероятный префикс на последнем шаге
            времени
111     return R_prev[0]
```

5.6 Dockerfile контейнера для обучения модели

Листинг 5.2

Dockerfile контейнера для обучения модели

```
1 # Базовый образ
2 FROM nvidia/cuda:9.0-cudnn7-devel-ubuntu16.04
3
4
5 # >> Начало Установка основного ПО
6
7 RUN apt-get update && apt-get install -y --no-install-recommends \
```

```
8         build-essential \
9         curl \
10        wget \
11        git \
12        python \
13        python-dev \
14        python-pip \
15        python-wheel \
16        python-numpy \
17        libcurl3-dev  \
18        ca-certificates \
19        gcc \
20        sox \
21        libsox-fmt-mp3 \
22        htop \
23        nano \
24        swig \
25        cmake \
26        libboost-all-dev \
27        zlib1g-dev \
28        libbz2-dev \
29        liblzma-dev \
30        locales \
31        pkg-config \
32        libsox-dev
33
34
35
36
37 # Установка Bazel
38 RUN apt-get install -y openjdk-8-jdk
39 RUN apt-get install -y --no-install-recommends bash-completion g++
       zlib1g-dev
```

```
40 RUN curl -LO "https://github.com/bazelbuild/bazel/releases/
     download/0.11.1/bazel_0.11.1-linux-x86_64.deb"
41 RUN dpkg -i bazel_*.deb
42
43 # Установка CUDA CLI Tools
44 RUN apt-get install -y cuda-command-line-tools-9-0
45
46 # Установка pip
47 RUN wget https://bootstrap.pypa.io/get-pip.py && \
48     python get-pip.py && \
49     rm get-pip.py
50
51 # << Конец Установка основного ПО
52
53 # >> Начало Конфигурация сборки TensorFlow
54
55 # Клонирование модифицированной версии TensorFlow из репозитория
     Mozilla
56 RUN git clone https://github.com/mozilla/tensorflow/
57 WORKDIR /tensorflow
58 RUN git checkout r1.6
59
60
61 # Настройка переменных окружения для GPU
62 ENV TF_NEED_CUDA 1
63 ENV CUDA_TOOLKIT_PATH /usr/local/cuda
64 ENV CUDA_PKG_VERSION 9-0=9.0.176-1
65 ENV CUDA_VERSION 9.0.176
66 ENV TF_CUDA_VERSION 9.0
67 ENV TF_CUDNN_VERSION 7.1.1
68 ENV CUDNN_INSTALL_PATH /usr/lib/x86_64-linux-gnu/
69 ENV TF_CUDA_COMPUTE_CAPABILITIES 6.0
70
71 # Настройка общих переменных окружения
```

```
72 ENV TF_BUILD_CONTAINER_TYPE GPU
73 ENV TF_BUILD_OPTIONS OPT
74 ENV TF_BUILD_DISABLE_GCP 1
75 ENV TF_BUILD_ENABLE_XLA 0
76 ENV TF_BUILD_PYTHON_VERSION PYTHON2
77 ENV TF_BUILD_IS_OPT OPT
78 ENV TF_BUILD_IS_PIP PIP
79
80 # Другие параметры
81 ENV CC_OPT_FLAGS -mavx -mavx2 -msse4.1 -msse4.2 -mfma
82 ENV TF_NEED_GCP 0
83 ENV TF_NEED_HDFS 0
84 ENV TF_NEED_JEMALLOC 1
85 ENV TF_NEED_OPENCL 0
86 ENV TF_CUDA_CLANG 0
87 ENV TF_NEED_MKL 0
88 ENV TF_ENABLE_XLA 0
89 ENV PYTHON_BIN_PATH /usr/bin/python2.7
90 ENV PYTHON_LIB_PATH /usr/lib/python2.7/dist-packages
91
92 # << Конец Конфигурация сборки TensorFlow
93
94 # >> Начало Настройка Bazel
95 RUN echo "startup --batch" >>/etc/bazel.bazelrc
96 RUN echo "build --spawn_strategy=standalone --genrule_strategy=
       standalone" \
97     >>/etc/bazel.bazelrc
98 RUN ln -s /usr/local/cuda/lib64/stubs/libcuda.so /usr/local/cuda/
       lib64/stubs/libcuda.so.1
99 RUN cp /usr/include/cudnn.h /usr/local/cuda/include/cudnn.h
100 ENV LD_LIBRARY_PATH $LD_LIBRARY_PATH:/usr/local/cuda/extras/CUPTI/
       lib64:/usr/local/cuda/lib64:/usr/lib/x86_64-linux-gnu/:/usr/
       local/cuda/lib64/stubs/
101
```

```
102 # << Конец Настройка Bazel
103
104 # Копирование текущей директории в контейнер в папку /DeepSpeech
105 COPY . /DeepSpeech/
106
107 WORKDIR /DeepSpeech
108
109 RUN pip --no-cache-dir install -r requirements.txt
110
111 RUN ln -s /DeepSpeech/native_client /tensorflow
112
113 # >> Начало Построение
114
115 WORKDIR /tensorflow
116
117 # Используемые оптимизации ЦПУ:
118 # -mtune=generic -march=x86-64 -msse -msse2 -msse3 -msse4.1 -msse4
      .2 -mavx.
119 # Добавлен флаг --config=cuda для сборки с использованием CUDA.
120
121 # Сборка префиксного декодера
122 RUN bazel build -c opt --copt=-O3 --copt="-D_GLIBCXX_USE_CXX11_ABI
      =0" --copt=-mtune=generic --copt=-march=x86-64 --copt=-msse --
      copt=-msse2 --copt=-msse3 --copt=-msse4.1 --copt=-msse4.2 --
      copt=-mavx //native_client:libctc_decoder_with_kenlm.so --
      verbose_failures --action_env=LD_LIBRARY_PATH=${LD_LIBRARY_PATH
      }
123
124 # Сборка DeepSpeech
125 RUN bazel build --config=monolithic --config=cuda -c opt --copt=-
      O3 --copt="-D_GLIBCXX_USE_CXX11_ABI=0" --copt=-mtune=generic --
      copt=-march=x86-64 --copt=-msse --copt=-msse2 --copt=-msse3 --
      copt=-msse4.1 --copt=-msse4.2 --copt=-mavx --copt=-fvisibility=
      hidden //native_client:libdeepspeech.so //native_client:
```

```
    deepspeech_utils //native_client:generate_trie --
    verbose_failures --action_env=LD_LIBRARY_PATH=${LD_LIBRARY_PATH
    }
126
127 # Сборка pip пакета TensorFlow
128 RUN bazel build --config=opt --config=cuda --copt=-mtune=generic
    --copt=-march=x86-64 --copt=-msse --copt=-msse2 --copt=-msse3
    --copt=-msse4.1 --copt=-msse4.2 --copt=-mavx //tensorflow/tools
    /pip_package:build_pip_package --verbose_failures --action_env=
    LD_LIBRARY_PATH=${LD_LIBRARY_PATH}
129
130 RUN ./configure
131 RUN bazel-bin/tensorflow/tools/pip_package/build_pip_package /tmp/
    tensorflow_pkg
132
133 # Установка TensorFlow
134 RUN pip install /tmp/tensorflow_pkg/*.whl
135
136 RUN cp /tensorflow/bazel-bin/native_client/
    libctc_decoder_with_kenlm.so /DeepSpeech/native_client/ \
137     && cp /tensorflow/bazel-bin/native_client/generate_trie /
        DeepSpeech/native_client/ \
138     && cp /tensorflow/bazel-bin/native_client/libdeepspeech.so /
        DeepSpeech/native_client/ \
139     && cp /tensorflow/bazel-bin/native_client/libdeepspeech_utils.
        so /DeepSpeech/native_client/
140
141 ENV TFDIR /tensorflow
142 WORKDIR /DeepSpeech/native_client
143 RUN make deepspeech
144 RUN make bindings
145 RUN pip install dist/deepspeech*
146
147 # << Конец Построение
```

```
148
149 # Использовать кодировку utf-8 для ввода/вывода Python
150 ENV PYTHONIOENCODING UTF-8
151
152 # Сборка инструментария KenLM
153 WORKDIR /DeepSpeech/native_client
154 RUN rm -rf kenlm \
155     && git clone https://github.com/kpu/kenlm && cd kenlm \
156     && mkdir -p build \
157     && cd build \
158     && cmake .. \
159     && make -j 4
160
161
162 # Готово
163 WORKDIR /DeepSpeech
```

5.7 Реализация модуля распознавания для системы поиска по речи в коллекции медиафайлов

Листинг 5.3

Реализация модуля распознавания для системы поиска в коллекции медиафайлов на языке Python 2.7

```
1 import os
2 import sys
3 reload(sys)
4
5 sys.setdefaultencoding("utf-8")
6
7 current_dir_path = os.path.dirname(os.path.realpath(__file__))
8 project_root_path = os.path.join(current_dir_path, os.pardir, os.
    pardir)
```

```
sys.path.insert(0, project_root_path)

import subprocess
import wave
import shutil
import const
from utils.yt_utils import download_yt_audio
from utils.slicing_utils import slice_audio_by_silence
from utils import audio_utils
from models import Transcription
from utils import db_util
from timeit import default_timer as timer
import file_transcriber
import indexer


# идентификаторы видео YouTube
YT_VIDEOS_TO_TRANSCRIBE = [
    "EC9t2-Lkgl4",
    "AhZa26nDZfA",
    "NTqB5ged4Rw",
    "SAQFzYnRTts",
    "LR_n8at2ORg",
    "LrHIBkjOl2Y",
    "0XRUbnKznOI",
    "uuULi6X6yqU",
    "RedxkKdFfkY",
    "6KiAr8w6o7E"
]

# функция проверки присутствия необходимых зависимостей
def check_dependencies():
    try:
```

```
42          subprocess.check_output(['soxi'], stderr=subprocess.STDOUT
                )
43          subprocess.check_output(['ffmpeg', '--help'], stderr=
                subprocess.STDOUT)
44      except Exception as ex:
45          print 'ERROR: some of dependencies are not installed:
                ffmpeg or sox: '+str(ex)
46          return False
47      return True
48
49  # функция обработки видео
50  def process_video(yt_video_id):
51      print("Обработка видео %s" % (yt_video_id))
52
53      video_data_path = os.path.join(const.VIDEO_DATA_DIR,
            yt_video_id)
54
55      print("Загрузка аудио...")
56      original_audio_path = download_yt_audio(yt_video_id)
57
58      print("Конвертирование аудио в формат wav...")
59      wav_audio_path = os.path.join(video_data_path, "audio.wav")
60      if not os.path.exists(wav_audio_path):
61          audio_utils.convert_to_wav(original_audio_path,
                wav_audio_path)
62
63      wav_vol_corr_path = os.path.join(video_data_path, "
            audio_vol_corr.wav")
64      print("Нормализация громкости")
65      if not os.path.exists(wav_vol_corr_path):
66          audio_utils.correct_volume(wav_audio_path,
                wav_vol_corr_path, db=-10)
67
```

```
68      wav_filtered_path = os.path.join(video_data_path, "
           audio_filtered.wav")
69      if not os.path.exists(wav_filtered_path):
70          print("Приминение полосного частотного фильтра...")
71          audio_utils.apply_bandpass_filter(wav_vol_corr_path,
               wav_filtered_path, low=2500)
72
73      wave_o = wave.open(wav_filtered_path, "r")
74
75      print("Разделение аудио по тишине...")
76      pieces, avg_len_sec = slice_audio_by_silence(wave_o)
77      print("Всего отрывков: %i, средняя длина открывка: %f" % (len(
           pieces), avg_len_sec))
78
79
80      pieces_folder_path = os.path.join(video_data_path, "pieces/")
81      if not os.path.exists(pieces_folder_path):
82          os.makedirs(pieces_folder_path)
83
84
85      print("Начало распознавания...")
86      for i, piece in enumerate(pieces):
87
88          piece_procesing_path = os.path.join(pieces_folder_path, "
               piece_%i_%.2f_processing.wav" % (i, piece["start"]))
89          piece_done_path = os.path.join(pieces_folder_path, "piece_
               %i_%.2f_done.wav" % (i, piece["start"]))
90
91          if not (os.path.exists(piece_procesing_path) or os.path.
               exists(piece_done_path)):
92              audio_utils.save_wave_samples_to_file(piece["samples"
                   ], n_channels=1, byte_width=2, sample_rate=16000,
                   file_path=piece_procesing_path)
93
```

```
94          if not os.path.exists(piece_done_path):
95              start_t = timer()
96              print("Распознавание отрывка %s" %
                    piece_procesing_path)
97              transcript = file_transcriber.transcribe_file(
                    piece_procesing_path).decode("utf-8").strip()
98              print("Распознавание заняло %.f секунд" % (timer()-
                    start_t))
99
100             print(transcript)
101
102             t = Transcription(media_type="youtube", media_id=
                    yt_video_id, time_start=piece["start"], time_end=
                    piece["end"], transcription=transcript)
103             # добавление распознанного текста с привязкой ко време
                    ни и видео в БД
104             db_util.add_item(t)
105
106             os.rename(piece_procesing_path, piece_done_path)
107
108
109 if __name__ == "__main__":
110     if check_dependencies():
111
112         db_util.init_db()
113
114         for yt_video_id in YT_VIDEOS_TO_TRANSCRIBE:
115             process_video(yt_video_id)
116             indexer.index_all()
```

5.8 Реализация модуля индексации для системы поиска по речи в коллекции медиафайлов

Листинг 5.4

Реализация модуля индексации для системы поиска в коллекции медиафайлов на языке Python 2.7

```
1  # =* coding: utf-8 *=
2
3  import os
4  import sys
5  reload(sys)
6  sys.setdefaultencoding("utf-8")
7  current_dir_path = os.path.dirname(os.path.realpath(__file__))
8  project_root_path = os.path.join(current_dir_path, os.pardir, os.
       pardir)
9  sys.path.insert(0, project_root_path)
10 import const
11 from whoosh.fields import Schema, TEXT, ID
12 from whoosh.index import create_in
13 from whoosh.index import open_dir
14 from whoosh.qparser import QueryParser
15 import Stemmer
16 from whoosh.analysis import StemmingAnalyzer, LanguageAnalyzer
17 from whoosh.lang.snowball import RussianStemmer
18 from utils import db_util
19
20
21 # функция индексации всех отрывков в БД
22 def index_all():
23     db_util.init_db()
24
25     # анализатор языка, включающий фильтрацию стоп-слов и стемминг
26     analyzer = LanguageAnalyzer('russian')
```

```
27
28      # определение структуры поискового индекса
29      schema = Schema(transcription_id=ID(stored=True),  transcript=
            TEXT(stored=True, analyzer=analyzer))
30      if not os.path.exists(const.TRANSCRIBED_WHOOSH_INDEX_DIR_PATH)
            :
31          os.makedirs(const.TRANSCRIBED_WHOOSH_INDEX_DIR_PATH)
32
33      # создание поискового индекса
34      ix = create_in(const.TRANSCRIBED_WHOOSH_INDEX_DIR_PATH, schema
            )
35
36      writer = ix.writer()
37
38      # индексирование всех распознанных отрыков в БД
39      for item in db_util.get_all_items():
40          writer.add_document(transcription_id=str(item.id).decode('
                utf-8'), transcript=item.transcription)
41
42      # фиксация индекса
43      writer.commit()
44
45  # функция поиска по индексу
46  def full_text_search(q):
47      ix = open_dir(const.TRANSCRIBED_WHOOSH_INDEX_DIR_PATH)
48
49      parser = QueryParser("transcript", ix.schema)
50      q = q.decode('utf-8')
51      query = parser.parse(q)
52
53      results = []
54
55      with ix.searcher() as searcher:
56          res = searcher.search(query)
```

```
57          for r in res:
58              results.append((
59                  r.fields()["transcription_id"],
60                  r.highlights("transcript")
61                  ))
62
63      return results
```

5.9 Реализация модуля поиска для системы поиска по речи в коллекции медиафайлов

Листинг 5.5

Реализация серверной части модуля поиска для системы поиска в коллекции медиафайлов на языке Python 2.7

```
1  # =* coding: utf-8 *=
2  import os
3  import sys
4  reload(sys)
5  sys.setdefaultencoding("utf-8")
6  current_dir_path = os.path.dirname(os.path.realpath(__file__))
7  project_root_path = os.path.join(current_dir_path, os.pardir, os.
       pardir)
8  sys.path.insert(0, project_root_path)
9  import indexer
10 from utils import db_util
11 import time
12
13 from flask import Flask, render_template, request,
       send_from_directory
14 from werkzeug.utils import secure_filename
15 import jinja2
16 import file_transcriber
```

```
17 import json
18
19
20 templates_folder_path = os.path.join(current_dir_path, "flask", "
     templates")
21 js_folder_path = os.path.join(templates_folder_path, "js")
22 tmp_folder_path = os.path.join(project_root_path, "tmp")
23
24 app = Flask(__name__)
25 app.jinja_loader = jinja2.FileSystemLoader(templates_folder_path)
26
27
28 # определение маршрутов Flask
29 @app.route('/js/<path:path>')
30 def send_js(path):
31     return send_from_directory(js_folder_path, path)
32
33 @app.route("/search")
34 def search():
35     return render_template("search.html")
36
37 @app.route('/search_results')
38 def get_search_results():
39     query = request.args.get('q', '')
40     return json.dumps(search(query), indent=3, ensure_ascii=False)
41
42
43
44
45 # функция поиска
46 def search(q):
47     db_util.init_db()
48     results_all = []
49
```

```
50      # поиск по индексу
51      indexer_results = indexer.full_text_search(q)
52
53      # извлечение необходимой информации из результатов поиска
54      for r in indexer_results:
55          db_item = db_util.get_item_by_id(r[0])
56          results_all.append({
57              "media_id": db_item.media_id,
58              "time_start": time.strftime('%Mm%Ss', time.gmtime(
                    db_item.time_start)),
59              "time_end": time.strftime('%Mm%Ss', time.gmtime(
                    db_item.time_end)),
60              "highlight": r[1]
61              })
62
63      media_ids = set(map(lambda x:x["media_id"], results_all))
64      results_grouped_by_media = []
65
66      # группировка результатов по идентификатору медиафайла
67      for mid in media_ids:
68          results_grouped_by_media.append({
69              "media_id": mid,
70              "matches": [r for r in results_all if r["media_id"] ==
                    mid]
71              })
72
73      return results_grouped_by_media
74
75  if __name__ == "__main__":
76      # запуск веб-приложения Flask
77      app.run(host= '0.0.0.0', port=5000, debug=True)
```

Листинг 5.6

Реализация клиентской части модуля поиска для системы поиска в коллекции медиафайлов на языке текстовой разметки HTML5 с вставками JavaScript

```
<html>
    <head>
        <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"
           ></script>
        <script src="js/lib/lodash.js"></script>
        <script src="https://unpkg.com/axios/dist/axios.min.js"></
           script>
    </head>

    <body>
        <div id="app">
          <div class="top-spacing"></div>
          <div class="logo-title"><b>deepspeech</b> search</div>
          <div class="search-form">
            <input class="search-query-input" v-model="query" v-on
               :keyup="onQueryChange">
          </div>
          <div class="results-container" v-for="result in
             search_results">
            <div class="result-card">
              <a v-bind:href="'https://youtu.be/'+result.
                 media_id+'?t='+result.matches[0].time_start"
                 target="_blank">
                <div class="card-left">
                  <div class="card-picture" v-bind:style="{ '
                     background-image': 'url(https://img.youtube
                     .com/vi/' + result.media_id + '/default.jpg
                     )' }"></div>
                </div>
              </a>
              <div class="card-right">
                <a v-bind:href="'https://youtu.be/'+result.
                   media_id+'?t='+result.matches[0].time_start"
                   target="_blank">
```

```
24                  <div class="card-title">${result.media_id}</
                        div>
25                </a>
26                <div class="card-time-marks" v-for="match in
                     result.matches">
27                  <a v-bind:href="'https://youtu.be/'+result.
                        media_id+'?t='+match.time_start" target="
                        _blank">
28                    <div class="card-time-match" v-html="match.
                          highlight"></div>
29                  </a>
30                </div>
31              </div>
32            </div>
33          </div>
34      </div>
35  </body>
36  <script type="text/javascript">
37      var app = new Vue({
38          delimiters:['${', '}'],
39          el: '#app',
40          data: {
41              query: '',
42              search_results: []
43          },
44          methods: {
45              onQueryChange: _.debounce(function (e) {
46                search(app.query)
47              }, 100)
48          },
49          created: function () {
50            console.log("mounted")
51            if(this.query != ""){
52              search(this.query)
```

```
53                }
54              }
55          })
56
57          function search(q){
58            console.log("search for "+q)
59            axios.get("/search_results?q="+q).then(function (
                response) {
60              app.search_results = response.data
61              console.log(response.data)
62            })
63          }
64
65      </script>
66  </html>
```